TÉCNICO LISBOA

A Reinforcement learning approach for the circle agent of Geometry Friends

João Luís Lopes Quitério

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Rui Filipe Fernandes Prada
Prof. Francisco António Chaves Saraiva de Melo

Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Rui Filipe Fernandes Prada
Member of the Committee: Prof. Luis Moniz

June 2015

Acknowledgements

First and foremost, I would like to thank my two supervisors, Professors Francisco Melo and Rui Prada, for their guidance and feedback since the moment I first started working with Geometry Friends. Without their precious inputs, I wouldn't have overcame most of the challenges this work imposed.

Secondly, I would like to leave a mention to all the professors of Instituto Superior Técnico with whom I had the pleasure to have lectures with. Without their huge expertise on their respective fields of study, I wouldn't have the knowledge needed to face the most important technological challenges of our days.

I also would like to thank the colleges, with whom I spent these five years and a half of my life, for all the fun and the discussions about how to find the optimal solutions for the problems that we were challenged to solve.

To the IEEE-IST Student Branch, NAPE-Taguspark and TEDx ISTTagus 2012 teams I would also like to address my gratitude. With them, I worked together with amazing people, I taught what I know to my fellow university colleges, I organized events for the community with some of the greatest experts of the country and I shared my passion for engineering among younger students.

Last, and definitely not the least, I would like to say a big thank you to my family and to my closest friends, that were always there to support me during this very important phase of my life.

Abstract

Geometry Friends (GF) is a physics-based platform game, which was part of the Artificial Intelligence (AI) competitions of the IEEE CIG Conference in 2013 and 2014. On GF there are two different characters, a circle and a rectangle, whose goal is to catch all the diamond-shaped collectibles available on each level of the game. In this work, a novel approach to the GF problem for the circle agent is proposed. This approach is based on learning algorithms, is character-agnostic and circumvents the excessive specialization to the public levels observed in the agents submitted to the 2014 competition. The solution uses a Divide-and-Conquer strategy that partitions the problem of solving a GF level into a series of three sub-problems: solving one platform (SP1), deciding the next platform (SP2) and moving from one platform to another (SP3). This method uses reinforcement learning to solve SP1 and SP3 and a depth-first search to solve SP2. To measure the quality of the developed agent, its results on the levels of the 2014 Competition are measured against the performance of that competition contestants, CIBot and KUAS-IS Lab. The results show that despite having a worse performance overall, the agent successfully avoided becoming over-specialized to a specific sub-set of levels.

Keywords: Agents, Geometry Friends, Over-Specialization, Divide-and-Conquer

Resumo

O *Geometry Friends* (GF) é um jogo de plataformas, com um motor de simulação de física, que fez parte do lote de competições de Inteligência Artificial das edições de 2013 e 2014 da IEEE CIG Conference. Neste jogo existem duas personagens, um círculo e um retângulo, que têm de colecionar todos os diamantes existentens num nível. Neste trabalho é proposta uma nova abordagem para o agente que controla o círculo. Esta abordagem é baseada em algoritmos de aprendizagem, é agnóstica relativamente à personagem e procura resolver o problema da especialização excessiva do agente aos níveis públicos verificada nos participantes da edição de 2014 da competição. A solução utiliza uma estratégia de Dividir para Conquistar que particiona o problema de resolver um nível do GF numa série de três sub-problemas: resolver uma plataforma (SP1); decidir qual a próxima plataforma para a qual o agente se deve mover (SP2); mover a personagem de uma plataforma para outra (SP3). Este método utiliza Aprendizagem por Reforço para resolver os problemas SP1 e SP3 e uma Procura em Profundidade Primeiro para resolver SP2. O agente implementado foi avaliado através da comparação de resultados obtidos nos níveis da competição de 2014 face aos obtidos pelos participantes. Os resultados mostram que, embora o agente tenha tido um pior resultado no geral, conseguiu evitar a especialização excessiva num conjunto de níveis.

Palavras-chave: Agentes, Geometry Friends, Especialização excessiva, Dividir para Conquistar

Contents

List of Figures

List of Tables

Nomenclature

Agent α Name of the agent developed in this work.

AI Artificial Intelligence

CIG Computational Intelligence and Games

DFS Depth-First Search

GF Geometry Friends

IEEE Institute of Electric and Electronics Engineers

MCTS Monte-Carlo Tree Search

Chapter 1

Introduction

1.1 Context

Geometry Friends (GF) is a physics-based platform game involving two characters: the Circle and the Rectangle. To complete the game, the two characters must overcome a number of levels. In each of those levels, the character must collect all the diamond-shaped objects in the environment in the fastest time possible. The layout of the levels can be designed so that they can be played by a single character or by both characters simultaneously, in a cooperative fashion. GF imposes challenges both in the single characters' movements and path planning through the levels and in the coordination of movements of both characters when cooperation is needed.

Due to the difficulty and variety of the challenges it imposes, GF is a good platform for developing new Artificial Intelligence (AI) algorithms. For this reason, GF has been featured in the game AI competitions of the IEEE CIG Conference in both 2013 and 2014. On both editions, there were two sets of levels, one that was made public during submission time and another that remained unknown until the competition. Existing AI solutions for GF, which were presented in past editions of the competition, were able to successfully tackle the public levels. Those solutions are mainly based on path planning algorithms. However, their performance in the unknown levels was significantly worse (approximately 25% less diamonds collected on the unknown levels), suggesting that those solutions were over-specialized in the levels that were made available. Hence, an AI system that is designed to successfully solve previously unknown GF levels, should be able to break down each new level in its simplest components and then control the agent in each one of these components. Another solution for the GF problem was proposed on a thesis that was written before the first edition of the AI Competition. However, this solution only tackled the problem of coordination of movements of the characters on the cooperative mode and so has no reported results on a single agent environment.

This work presents a new approach to the problem of solving GF for the circle agent. This approach splits a GF level into three simpler problems: solving a single platform, choosing the next platform to move to and moving from one platform to the other. Since these three problems can be presented to both circle and rectangle characters, the solution is character-agnostic and can be implemented in the

agent that controls the rectangle in the future. To tackle each one of the sub-problems, the agent makes use of either reinforcement learning or path planning algorithms, depending on the problem it is solving.

The implemented agent ran on a training set of twenty-eight different levels, with five of them being the public levels of the 2014 GF Competition. To evaluate the quality of the implemented agent, it ran under the same rules and on the same set of levels of that competition. The results were compared against the performance of both 2014 GF Circle Agent Competition contestants: CIBot and KUAS-IS Lab agents.

1.2 Dissertation Goal

The goal of this work is to have a solution for GF that overcomes the over-specialization to the public levels observed in the other solutions. To successfully achieve this goal, the implemented agent must:

- Perform better than the winner of the IEEE CIG 2014 GF AI Competition, CIBot on the private levels of that competition;
- Collect an average of more than 50% of the diamonds and have a difference between the percentage of diamonds caught in the private and on public levels sets smaller than 10%.

Both these goals must be achieved without the agent having previously trained on any of the private levels. Another goal of this work is to develop an agent that can be used both as a benchmark for the Circle AI Competition and as springboard for future developments of new AI agents for GF.

1.3 Document Outline

This dissertation is divided into eight chapters. This first chapter gives the motivations and the goals for this work. Chapter 2 presents the details of the Geometry Friends game. In that chapter, the characters and the challenges imposed by the game are presented. Chapter 3 makes a review of the state of the art of research on AI in games and on the development of AI solutions for GF. Chapters 4 and 5 give the details about the solution implemented in this work. The former chapter explains how the GF problem was divided into three independent sub-problems and how each one of them is solved by the agent; the latter focuses in explaining the software architecture behind the implemented agent. Chapter 6 shows how the developed agent was trained before being subjected to the evaluation. On chapter 7, both the setup and the results of that evaluation are presented. Moreover, there is also a discussion about the pros and limitations of the solution proposed. Finally, on chapter 8, the conclusions of this dissertation are drawn and some future work on AI for GF is motivated.

Chapter 2

Geometry Friends

Geometry Friends (GF) is a two-player platform game that is set in a two-dimensional game space. The game uses the Farseer Physics Engine [1] to provide a simulated physics environment where gravity, mass and attrition affect the characters. The goal of GF is to catch all the diamond-shaped objects that are distributed throughout the game space in the fastest time possible.

There are two different characters in the game that can be controlled by the player: a yellow circle and a green rectangle. Each character has a specific set of actions that it can perform. The outcome of these actions can be affected by collisions with the other character. A depiction of the characters' possible movements is made in Figure 2.1. For the circle character, the following actions are available:

- *Roll* to the left;
- *Roll* to the right;
- *Jump*;
- *Change its size*.

The rectangle, on the other hand, can perform the following actions:

- *Slide* to the left;
- *Slide* to the right;
- *Morph* to become wider or slimmer, while maintaining a constant area.

The environment is populated with obstacles, which also constraint the mobility of the characters in the level and diamond-shaped objects that can be collected. There are two types of obstacles on GF:

- Black obstacles, that restrict the movement of both character;
- Coloured obstacles that only restrict the movement of the character of the opposite colour.

[1] `https://farseerphysics.codeplex.com/`

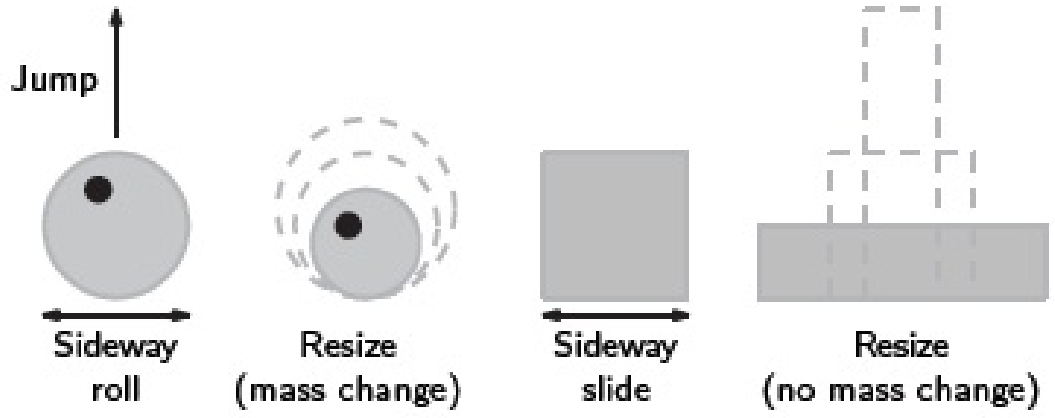


Figure 2.1: Possible movements of both characters. Taken from Geometry Friends AI Competition Website

Both these types of obstacles can be static, if their position is always the same, or dynamic, if their position changes over time.

GF has different levels, each one with a distinct layout for the platforms and the diamonds. Since each character has different motor skills, there are levels that be solved by only one character; levels that can be solved by any of the two; and levels that can only be solved by both agents acting cooperatively (*eg.* Figure 2.2). GF also features a level editor that can be used to create new levels or to edit the existing ones.

2.1 Geometry Friends Challenges

The GF environment imposes different kinds of challenges for both single agent and cooperative modes.

When considering only a single character's point-of-view, GF imposes challenges in the navigation within the level. When considering only a single agent scenario the major challenges are:

- Finding a path that maximizes the number of diamonds that can be collected;
- Being capable of recalculating that path whenever the agent fails to perform an action;
- Finding the action that gets the character closer to a target diamond;
- Avoiding getting into locations where the level becomes unsolvable;
- Being capable of performing an action in perfect timing, taking into account the character's speed and position;
- Being able to deal with the possible dynamism of the game environment (dynamic obstacles and/or the presence of the other character).

In the cooperative mode, there are challenges that are imposed by the need of achieving coordination of movements of both characters. To achieve such coordination, the agent needs to understand what

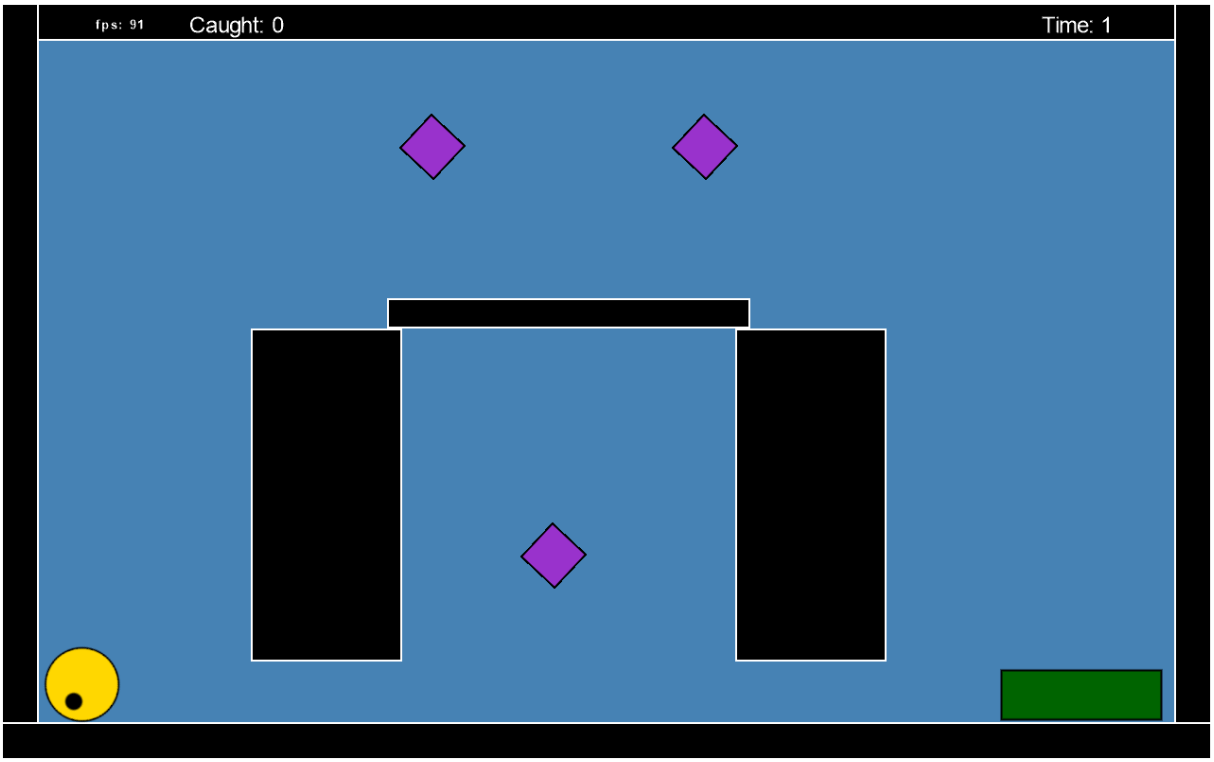


Figure 2.2: Example of a Geometry Friends level. To solve this level, the bottommost diamond can only be caught by the rectangle character. To get to the two top collectibles the circle has to use the rectangle as a springboard to reach the obstacle nearer those diamonds.

the other agent is going to do. Because there isn't any direct communication between the agents, either one uses the knowledge of the other's algorithm to bias its decisions, thus becoming bounded to that specific team-mate, or uses algorithms of plan and intention recognition to understand what actions the other agent is going to perform [33, 34, 35].

2.2 AI Competition

Geometry Friends is seen as a good platform to test new AI approaches due to the challenges it imposes. In order to motivate the development of new AI solutions, a Geometry Friends AI competition has been held at the IEEE CIG Conference since 2013. There are three distinct tracks is this competition: two single agent tracks (one for each one of the characters) and a cooperative track (where both agents play together). In each track, the agents taking part in the competition, face a set of ten different levels of GF. The ten levels of the competition are depicted on Appendix D. In the competition, each level has a maximum number of seconds that the agent can use to solve it. Out of those ten levels, five of them are made available to the public before the competition agent submission deadline and the other five are only disclosed once the competition finishes. For each level, the agent is ran ten times and its score over those runs is averaged when calculating the final score for that level.

According to the competition website[2], in the future there will be tracks for Human and AI Player

[2] http://gaips.inesc-id.pt/geometryfriends/

Cooperation – where autonomous agents will play the cooperative levels with a human-controlled teammate – and for Agent Believability – where agents are required to solve the level in a humanly believable fashion.

Chapter 3

Related work

This chapter makes a brief review of the state of the art of Ai in games. The main focus of this chapter goes to previous solutions for GF.

3.1 Usage of AI in games

The usage of AI to solve games is a long standing tradition. One of the oldest AI benchmarks games is Chess. Turing suggested that Chess could be solved using a MiniMax algorithm [1]. The computer program DEEPBLUE become the world's best chess player in 1997 when it beat the chess grandmaster Kasparov [2]. Other games have been used showcase the abilities of AI Systems. Some of the outstanding AI achievements were the performances obtained by CHINOOK in checkers [3], and WATSON in jeopardy [4].

There are several techniques that can be used in games, being finite state machines, the usage of scripting and the development of agents the most common [5]. Finite State Machines and Scripting are used in some of well-known games such as Bioware's Neverwinter Nights and Valve's Half-Life. The former consists on dividing each game's object behaviour in a set of logical states [6]. Hence, for each type of behaviour there is a state and a set of actions associated with it. Scripting, on its turn, simplifies the creation of a character's behaviour by hiding complex aspects of the a game [7]. Both Scripting and Finite State Machines techniques require that the character's behaviour is hard-coded and, therefore, deterministic.

Intelligent agents perceive the environment and act in order to achieve their goals. Games are a ideal stage to benchmark the performance of an AI Agent, because they provide environments where there is limited information and decisions must be taken under a certain time frame [5]. To promote the development of Intelligent Agents for games, there have been several AI Competitions in recent years. These competitions feature classical arcade-style games, such as Super Mario [8] and Ms.Pac-Man [9] or more complex environments such as the First-Person Shooter Unreal Tournament [10, 11] and StarCraft [12].

Since GF is a platform-based game, it is important to look at a similar platform – Super Mario. Robin

Baumgarten developed an agent that uses the A* search algorithm to find the quickest route that doesn't get Mario killed [13]. The agent leverage the fact that the game physics engine is open-source to build a simulation engine to predict future states of the game. The A* algorithm runs in a graph whose nodes are generated using the whole world-state at a given moment. The neighbouring nodes are given by the world-state one further step in the simulation. Other approaches to Super Mario include the usage of rule-based agents that store hard-coded condition-action rules [14] and agents that rely on genetic algorithms to improve the output of a Finite State Machine [15].

The usage of learning algorithms was also tested in Super Mario. Ross and Bagnell [16] use structured learning to develop a player for the Infinite Mario game. Tsay *et at* apply reinforcement leaning to an agent that plays Super Mario that outperforms other learning approaches [17]. However, it still under-performs when compared an search-based A* approach. Learning solutions are popular nowadays since the most recent successes on AI arise from the combination of powerful search algorithms with an equally powerful *learning component* [4, 18]. For example, recent results on computer Go rely heavily on Monte-Carlo tree search algorithms rooted in reinforcement learning, such as the UCT algorithm [19]. In a closely related line of work, the Deep-Q system combines reinforcement learning with a deep neural network to achieve human-level play of several Atari games [20].

In this work, reinforcement learning is combined with search as a means to learn policies that can be generalized across levels. RL algorithms enable an agent to learn a task by trial and error, driven by a reward signal that "encodes" the task to be learned [21]. RL methods are designed for situations where the environment is quite dynamic and non-deterministic, because agents that implement this type of learning mechanism have the capacity to learn from experience [22]. The reinforcement learning algorithms can be also be used to learn how to achieve coordination of movements when there is the need to cooperate [23, 24]. On reinforcement learning problems, the major challenge is to manage the trade off between the usage a coarse-grained manner to define the game states – which would lead to ignore several important situations on the game – or of a fine-grained state representation, which would lead to an enormous amount of states to be processed. Reinforcement learning can be implemented using utility-based agents, which already know the states of the game and only need to learn the utilities for them; reflex agents that build a policy that map a state to an action; and Q-Learning agents, that learn a action-utility function that returns the expected utility for an action on a given state [25].

3.2 AI in Geometry Friends

The related work on GF can be split into two different categories: how the game was designed and the AI techniques that were implemented in agents to control the game characters.

3.2.1 Geometry Friends design

GF was first mentioned by Rocha et al. [26] when it was still being developed. According to that paper, the original levels of GF are designed to force cooperation between both characters. It is assumed that

the game is to be played solely with human-controlled agents. The position of the diamonds is calculated so as to avoid situations where the characters can split the task of collecting the diamonds, thus solving the level with a divide-and-conquer approach. According to the article, GF imposes Coordination, Reflexion/Action and Spatial-Awareness challenges as they are natural challenges on platform-based games.

On his master's thesis, Rocha [27] gives all the details about the design of Geometry Friends. On that work, the author states that the characters are to be controlled using the Nintendo Wii Controller. According to Rocha, the main characteristics of the game are:

- Cooperation – The characters have complementary abilities and so they must cooperate to overcome the obstacles of the game;
- The Wii Remote – It imposes challenges on the control of the characters as the intensity of the movements are bounded to the accelerometer reading taken by the controller;
- Physics-Based Gameplay – The game uses physical concepts such as gravity and attrition to create a different gameplay experience from the ones found on typical 2D Platform games.

3.2.2 Geometry Friends AI Systems

The development of AI for Geometry Friends started with Carlos Fraga [28], who presented an AI solution to GF using a navigational graph (see Figure 3.1). The work done by Fraga is prior to the first edition of the AI Competition and only targets the cooperative environment of GF. The graph has different types of edges depending on whether the edge is traversable by the circle alone, by the rectangle alone, or by both characters. The nodes of the graph are positioned on both agents' starting positions and on the diamonds' positions. Other nodes are then generated by expanding those initial nodes. Every time the agent has to perform an action, it runs the A* algorithm to determine the path to follow. The author called the action of navigating through the path the agent has to follow a Task. After calculating the task to be done, the agent divides it into a series of actions. These actions are sets of movements that make the character reach a specific node of the graph that is on the path. After completing each action, the agent checks if it reached the planned action outcome. If it had done so, the agent repeatedly proceeds to next action on the task until that task is fulfilled. Whenever an action is unsuccessful or the task is finished, the agent calculates the next task to be done. One of the limitations of this approach is the processing overhead caused by running the A* algorithm every time the agent has to be calculate a task. Another problem is that each agent is only able to cooperate with another agent sharing the same algorithm, which imposes a significant limitation when playing with an arbitrary team-mate.

Since 2013, an annual GF AI Competition has been held and participants have been encouraged to write either a full-sized paper or a technical report about the approaches used to solve the GF problem.

Yoon and Kim [29], developers of the winning agent of the 2014 GF AI Competition circle track and Benoît *et al* [30], runner up in the 2014 rectangle track of the same GF competition use similar approaches based on path planning. Both agents create a graph from the level configuration and use

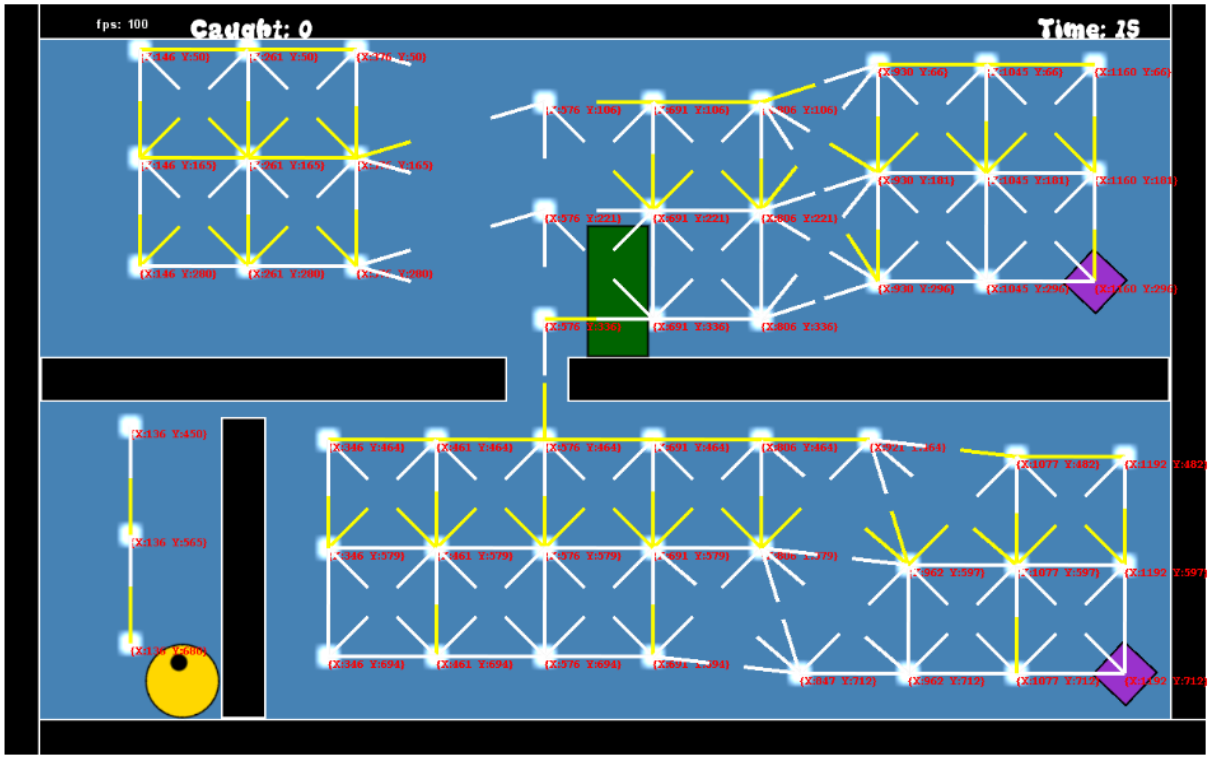


Figure 3.1: Fraga's Navigational Graph. The yellow edges are only traversable by the circle character, whereas the white ones can be traversed by both.

Dijkstra's algorithm to find the shortest path through the graph. Yoon and Kim's agent uses the edge points of the platforms as nodes (Figure 3.2). Whenever it is possible for the circle to go from one of those edge points to another a graph edge is created. Every time the agent has to play, it runs Dijkstra algorithm to find the shortest path to the closest diamond. To avoid constantly running the Dijkstra algorithm, the edge points along the path to a diamond are stored in a queue in the order they should be visited. With this optimization, the algorithm is only ran when the queue is empty. Benoît's agent firstly creates a meta-graph with special points called objective points. With this meta-graph, the agent calculates the order in which it must visit those points using Dijkstra's algorithm. Finally, the agent plans the set of actions it has to perform to be able to follow the path found. The main difference between those two solutions is the fact that Benoît's agents plan everything on a initial set-up phase, whereas Yoon's agents make plans while playing the level.

Kim and Kim [31], developers of the winning agent of the square track of the 2014 AI Competition, applied the Monte-Carlo Tree Search (MCTS) algorithm to find a path that allows the agent to catch all the diamonds of the level. On previous attempts, the authors relied solely on the A* algorithm to calculate the optimal path from the agent to one of the diamonds. However, they report that the A* approach fell into a local-optimal solution for the game. In the newly proposed approach, all the computation is done on an initial set-up phase. The agent builds a directed graph from the level layout, whose edges are assigned to one of three specific shapes of the agent:

- Square shape;

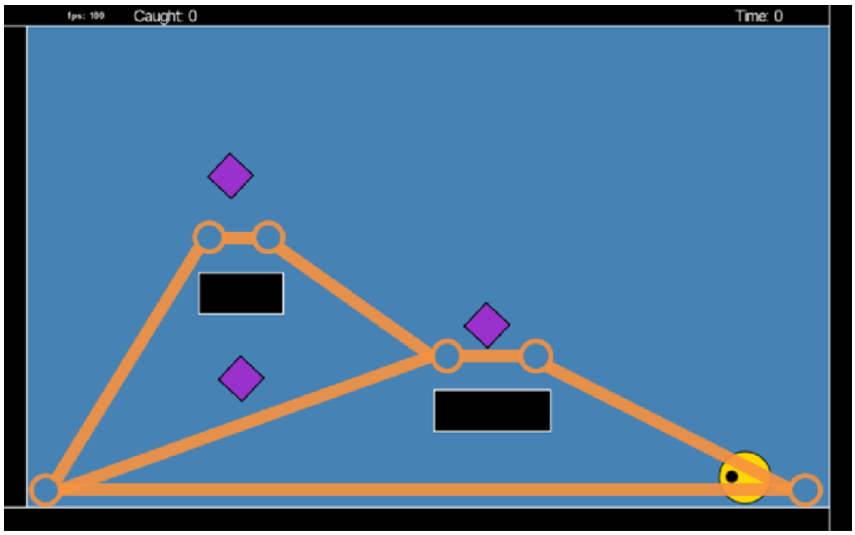


Figure 3.2: CIBot Circle Agent's Graph. The nodes of the graph are the edge points. Image taken from Yoon and Kim paper [29]

- Tall rectangular shape;
- Wide rectangular shape.

The graph is converted into a tree that can be traversed by the MCTS algorithm (Figure 3.3). The path that is returned by the MCTS can be the optimal path if it is allowed to be run enough time.

Finally, Yen-Wen Lin et al. [32] developed the KUAS-IS agent that also competed on the 2014 GF AI circle competition. Their agent uses A* and Q-Learning to solve GF. A* is used to find the shortest among the paths that go through all the diamonds on the level. The agent then relies on Q-Learning to learn how to move in that path. The feature vector used by this agent is composed of seven features that include the path and direction calculated with the A* Algorithm, the possible movements that the agent can perform and the circle radius. This work uses Q-Learning because this approach can learn how to avoid some of the pitfalls present in the GF levels.

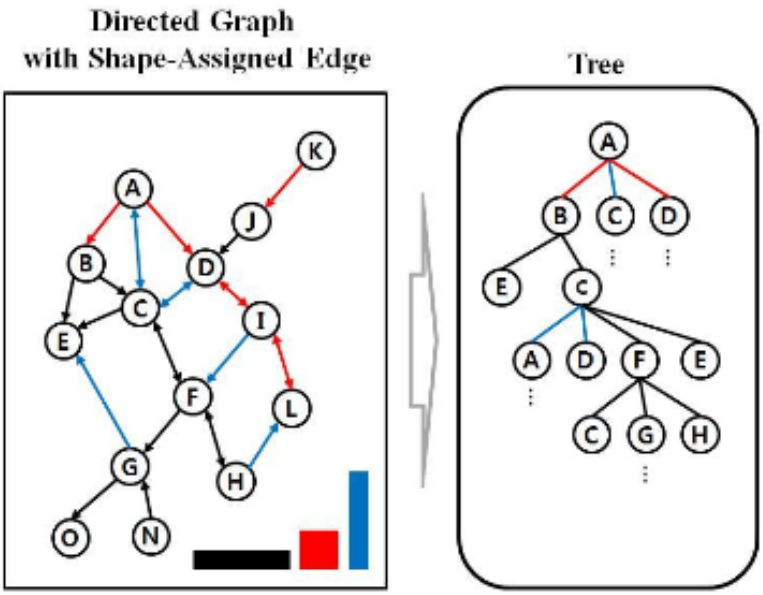


Figure 3.3: CIBot Square Agent's Graph to Tree conversion. The red edges are traversable when the character is in square shape; the blue edges are traversable when the character is in tall rectangular shape; black edges are traversable when the agent is in the wide rectangular shape. Image taken from Kim and Kim paper [31]

Chapter 4

Solving a Geometry Friends Level

As discussed in Section 1.2, the main goal of this work is to overcome the over-specialization to the public levels of GF observed in previous AI systems. In order to achieve the necessary generalization ability, it is crucial that the agent identifies and solves simple components of the level that are repeated throughout the game, instead of focusing on level-specific situations. So as to increase the probability of finding those repeatable patterns, the agent uses a divide-and-conquer approach to the problem of solving a GF level, splitting the task into a set of three simpler sub-problems:

SP1 Catching all the diamonds that are on a platform;

SP2 Deciding the next platform to go;

SP3 Moving to the other platform.

Using this approach, solving a GF level is reduced to repeatedly solving series of ($SP1 \rightarrow SP2 \rightarrow SP3$), starting form the platform where the character is initial placed, until all the diamonds on the level are caught. The agent is going to look for repeatable patterns on SP1 and SP3, as SP2 always needs to take into account the level physical layout when deciding to which platform the character should move next.

To ensure that this approach works, a platform isn't always a mapped to a whole physical obstacle of the game. Instead, a platform must be defined as an area of the game space where the agent can roll from end to end without having to jump. In figure (4.1), it is possible to see that the ground platform is divided into two smaller platforms ($p2$ and $p3$) as obstacle $o1$ restricts the area that can be reached by the circle character. The generation of these platforms is made on an initial set-up phase of the game by an agent module called the Platform Manager. This module and the platform generation algorithm is discussed on detail in Section 5.1.3.

The diamonds available on the level are assigned to one of those platforms. The attribution of a platform to a diamond is made in a left to right and top to bottom order. A diamond is always assigned to the platform that is exactly bellow it. However, this algorithm can lead to diamonds being assigned to platforms from where it is impossible to collect them. This is a limitation of this solution that is discussed later in Section 7.3.

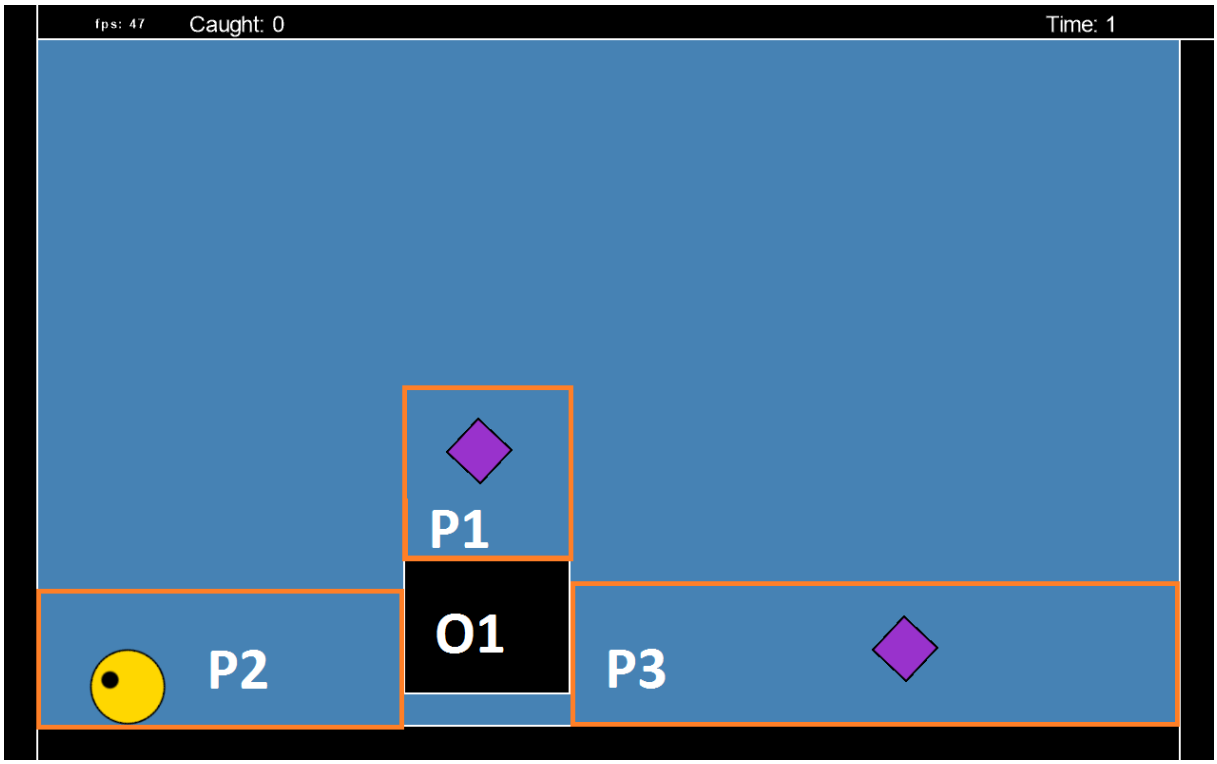


Figure 4.1: Platform division example. The ground platform is divided into platforms $P2$ and $P3$ as $O1$ restricts the movement of the circle in the ground platform.

Sub-problems SP1 and SP3 are solved using reinforcement learning. The agent is looking to find two distinct policies (one for each one of the sub-problems) that it can use to solve any possible instance of those problems. The use of a reinforcement learning algorithm is explained by the need to find a solution that can be applied to any level of GF regardless of the agent having previous knowledge of that level. To use reinforcement learning, one needs to define feature vectors that are used to calculate the reward of each game-state. SP1 and SP3 need different feature vectors as the problems differ very much from one another. To solve SP1 the agent only needs to capture features of the platform the character is in, whilst to solve SP3 it also must take into account the destination platform and the distances between the two platforms. Sections 4.1 and 4.1 discuss in detail the features used in SP1 and SP3 respectively.

SP2, however, is formulated as path planning problem similarly to what other approaches to GF have done. The goal of this problem is to find a path, starting on the character's current position, that goes through all platforms with diamonds to be caught. However, in this solution a Depth First Search is used instead of the A* algorithm used in other GF solutions. A DFS is used because:

- The search space is very small;
- The goal is to find a path that goes through all the diamonds on the level (doesn't have to be the optimal one);
- The DFS is efficient enough to be ran whenever the agent needs to recalculate its path;
- The DFS can be easily implemented and doesn't have a big memory overhead.

The details on SP2 are explained on section 4.2.

4.1 Solving one platform

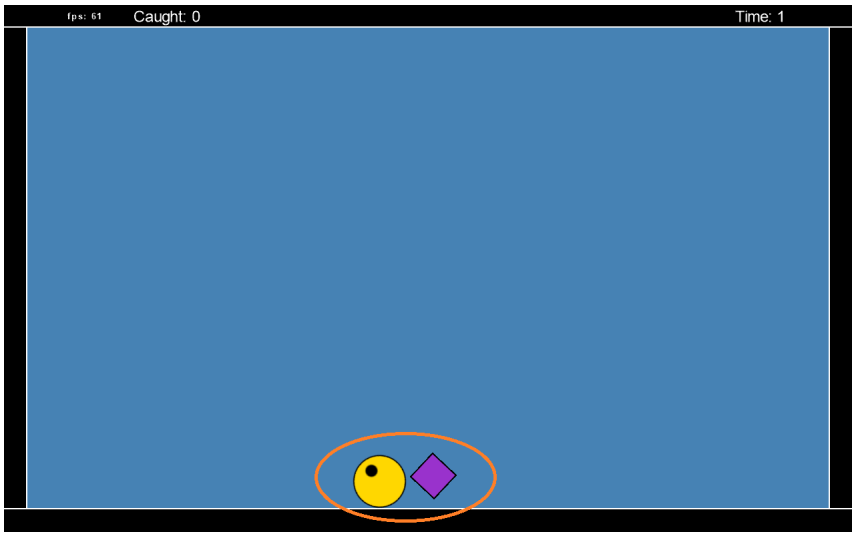
When solving one platform there are different approaches that can be used to collect the diamonds on it. Some of these approaches rely on intelligent algorithms to discover the fastest way to go through all the diamonds on that platform. Examples of such intelligent approaches can be seen on other solutions to the GF problem. The solution proposed in this work, however, collects the diamonds in a greedy way, taking only into account the diamond that is closer to the character. This assumption, even though fails to capture the fact that the character can fall off the platform without solving it completely, makes this sub-problem easier to define. As the agent is always looking to the closest diamond, the position of the other diamonds of the platform can be ignored from the feature-vector. Although this assumption may lead to sub-optimal solutions, as most of the times the agent will not follow the fastest path within the platform, it speeds up learning because, by only considering the closest diamond, the agent will more frequently find the same scenario. For example, the situations depicted on Figure 4.2 are perceived to be the same. To capture the relation of positions between the character and the diamond, the feature-vector stores the distance vector that goes from the character's current position to the position of that diamond. The coordinates of that vector are measured as the number of circle radius that are needed to fill the distance between the character and the diamond.

If the agent only used the distance vector to the closest diamond , it would become harder to track the number of available diamonds within the platform at a given time, which is important to calculate the reward for the state. Therefore, the feature vector also contains the number of diamonds that are still available on the platform. Another feature that is important to take into account is how close the platform is to a physical limitation of the level or to another obstacle. This is an important feature because the character can move against that obstacle and avoid falling off the platform. If the agent doesn't consider that feature, depending on the previous training, it can either have a careless behaviour and go at full speed on every diamond that is on the edge of the platform, falling off the platform every time there is no obstacle to avoid it; or become too careful and always go slowly to a diamond and lose precious time in situations where it could leverage the presence of an obstacle to avoid the fall. To capture such cases, the agent has a feature that indicates whether there is an obstacle that will prevent its fall from the platform. This feature is only considered when the distance from the closest diamond to the platform edge is less than five times the circle radius and there is such an obstacle. That is the distance the character needs to reverse the direction of its movement when it is rolling at its fastest speed (measured empirically).

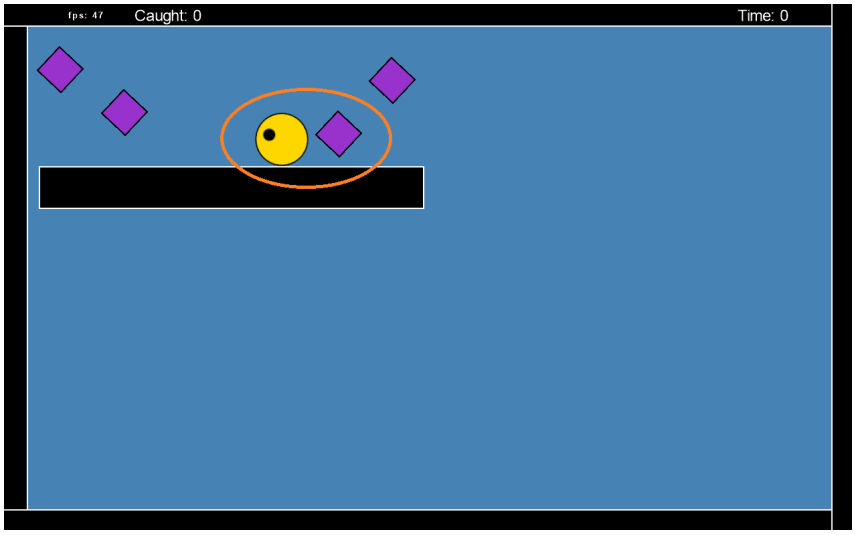
Another important feature to take into account in this problem is the position of the character within the platform. To capture this, there are three distinct possibilities:

- Considering both character's and platforms absolute positions in the level;
- Considering the position of the character relative to the platform;
- Considering the distances from the character's position to the left and right edges of the platform.

The first approach is the worst of the three because it is bounded to the game environment layout.



(a) Situation 1.



(b) Situation 2.

Figure 4.2: Generalization obtained by using a greedy approach. Even though the number and position of the diamonds on the platform is different, the agent perceives both situations as the same. The agent only takes into account the area enclosed by the orange ellipsis.

Hence, when two platforms have the same configuration but are located in different places of the level, the agent will fail to see the similarities. The second approach tackles that limitation. It does so by creating a new reference system for each platform and calculating the character's position according to that new reference system. The relative position of the character can be either calculated using the character's position coordinates within that platform's reference system or by calculating the fraction of the platform the agent is in (for example, if the character is at 50%, it means that it is in the middle of that platform). The former limitation is failing to capture the distance between the character's position and the limits of the platform, whilst the latter fails to capture the size of the platform as on a very tiny platform, even when the character is at 50% of that platform, a single move can make the character fall off it.

To tackle the first two approaches' limitations, the agent uses the distances from its character's

position to both edges of the platform to locate itself within that platform. This is the only solution, out of the three presented, that isn't bounded to the level configuration and that takes into account the size of the platform the character is in. This feature is also measured as the number of circle radii needed to fill the distance.

The horizontal speed of the character is also used as a feature. The speed value on the game is a real value that ranges from -200 to 200. In this solution that value is weighted by a 1/20 empiric constant and then truncated to an integer value to improve generalization. Hence, the speed value stored in the feature vector ranges from -10 to 10. The vertical speed is not taken into account since the agent can only perform movements when it is in contact with a platform, which is the same as saying that the character's vertical speed is 0.

Summarizing, this sub-problem's feature vector has the following values:

- Distance to the left edge of the platform;
- Distance to the right edge of the platform;
- Horizontal speed;
- Number of diamonds remaining on the platform;
- Distance vector to the closest diamond;
- Presence of a wall preventing falling from the platform.

To determine the reward for a state in this sub-problem it is important to look not only to the features of the state itself, but also to the impact that state had in the global performance of the agent in the level. Hence, to calculate such reward value, it is important to take into account the following features:

- The number of diamonds on the platform that were collected on the end of the level ($\#DiamondsFinal$). This is an indication of good was the outcome of that platform was. The more diamonds assigned to the platform that were collected in the end of the level, the better the platform performance was;
- The number of diamonds on the platform that were already collected by the time the agent was on that state ($\#DiamondsState$). The greater the number the better the state is. Together with the number of diamonds of the platform that were collected by the end of the level ($\#DiamondsFinal$), this feature indicates how close that specific state was from the one where the agent to collected the last of the diamonds on the platform.
- Distance to the closest diamond ($Distance$). The closer the agent is to the diamond better the state is.
- Percentage of time available ($TimeAvailable$). The more time the agent still has to solve the level the better the state is.

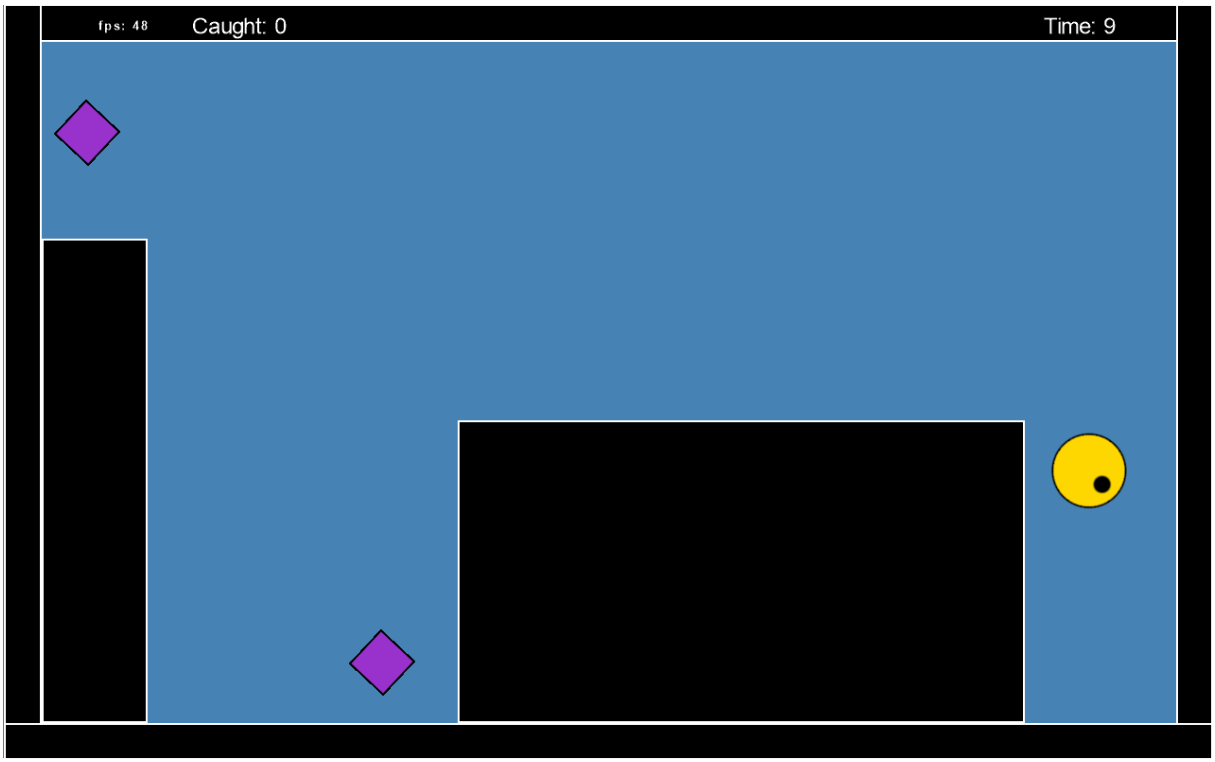


Figure 4.3: Example of a bad decision of the agent. In this level from GF IEEE CIG 2014 AI Circle Single Track Competition, the circle will not be able to catch any diamonds due to the bad decision of falling to its right.

The reward function for this sub-problem, for a certain state s, is given by Equation (4.1). The distance factor has a greater weight as it is the feature that has most importance on this sub-problem

$$\begin{aligned} reward(s) = & \#DiamondsFinal \\ & + \#DiamondsState \\ & + 10 \times \frac{1}{Distance} \\ & + TimeAvailable \end{aligned} \tag{4.1}$$

This game-state definition is only applied when the character is in a platform which still has diamonds to be caught. When this is not the case the agent needs to choose which platform the character should move to next.

4.2 Planing next platform

Making a good decision regarding the next platform to move to is crucial. In GF, some of the levels, like the one depicted on Figure 4.3, are designed in a way that a wrong move from the agent can jeopardize the success of that level.

To make those decisions, the agent runs a modified version of a Depth-First Search (DFS) in a navigation graph that is stored in the Platform Manager module. The DFS skips the validation on whether

the agent has already visited a node from the classic implementation. This is done because the agent may have to return to a previously visited node to get to other unvisited nodes. The nodes of the graph represent the platforms of the level. Each one of the nodes have a value assigned to it. That value represents the number of diamonds on the platform. The edges represent the existing connections between two platforms. A graph edge is created whenever it is possible, for the character controlled by the agent, to go from one platform to the other. The graph is a directed-graph because it can be possible for the character to go from one platform to another but not the other way around. This graph is generated on an initial set-up phase and uses an algorithm that is detailed on section 5.1.3. There can be more than one edge connecting a pair of nodes. This happens when it is possible to go to the other platform by more than one way, for example by falling to the left or by falling to the right of the platform the character is on. The edges also store the xx coordinate of the point where the edge has its start. Those specific points are called jump-points. The jump-points are used to distinguish the edges whenever there are more that one edge connecting the same pair of nodes. Another information stored in the edge structure, is the difference in both xx and yy axis between the jump-point and the destination platform. This distance is calculated using the point on the destination platform that is closer to the jump-point.

The DFS returns the path that maximizes the number of diamonds that can still be caught. The path is represented as an ordered list of nodes that have to be visited by the character. The search algorithm assumes that, for each platform the character visits, it can collect all the diamonds on it. Since the node is not being marked as visited, the DFS can degenerate in an infinite cycle, as the algorithm can get stuck in a situation where it goes back and forth on the same pair of nodes. To avoid being caught on such cycles, the DFS depth is limited to $2 \times (numberOfPlatforms - 1)$. This value ensures that, if a solution exists that solves all the level, it will be find. $2 \times (numberOfPlatforms - 1)$ is the number of movements the agent has to make to solve the edge case scenario where it has always to return to the start platform before visiting another platform.

After computing the path, the agent calculates which is the closest edge that goes to the first platform of the path calculated. The agent commits itself in moving to next platform through that selected edge. This committed is reviewed whenever the character leaves the platform it is in, whether because the agent performed a jump action or because the character fell off the platform. Once the agent lands on a platform again, and that platform doesn't have any diamonds to catch, it re-runs the DFS and re-commits itself to an edge. In certain cases, this newly committed edge can be the same as the one the agent was targeting before. This can happen when the agent fails to get to the intended platform and remains on the same platform it was before. The process of running the DFS and committing to an edge is repeated until the character reaches a platform where there are diamonds to be caught.

The usage of the DFS is efficient as the depth of the search is limited by the number of nodes on the graph. This fact enables the DFS to be ran every time the agent has to make a move and makes recalculation of the path a quick operation to execute. The performance of the DFS is very important as it must be used whenever a deviation from the original one is made.

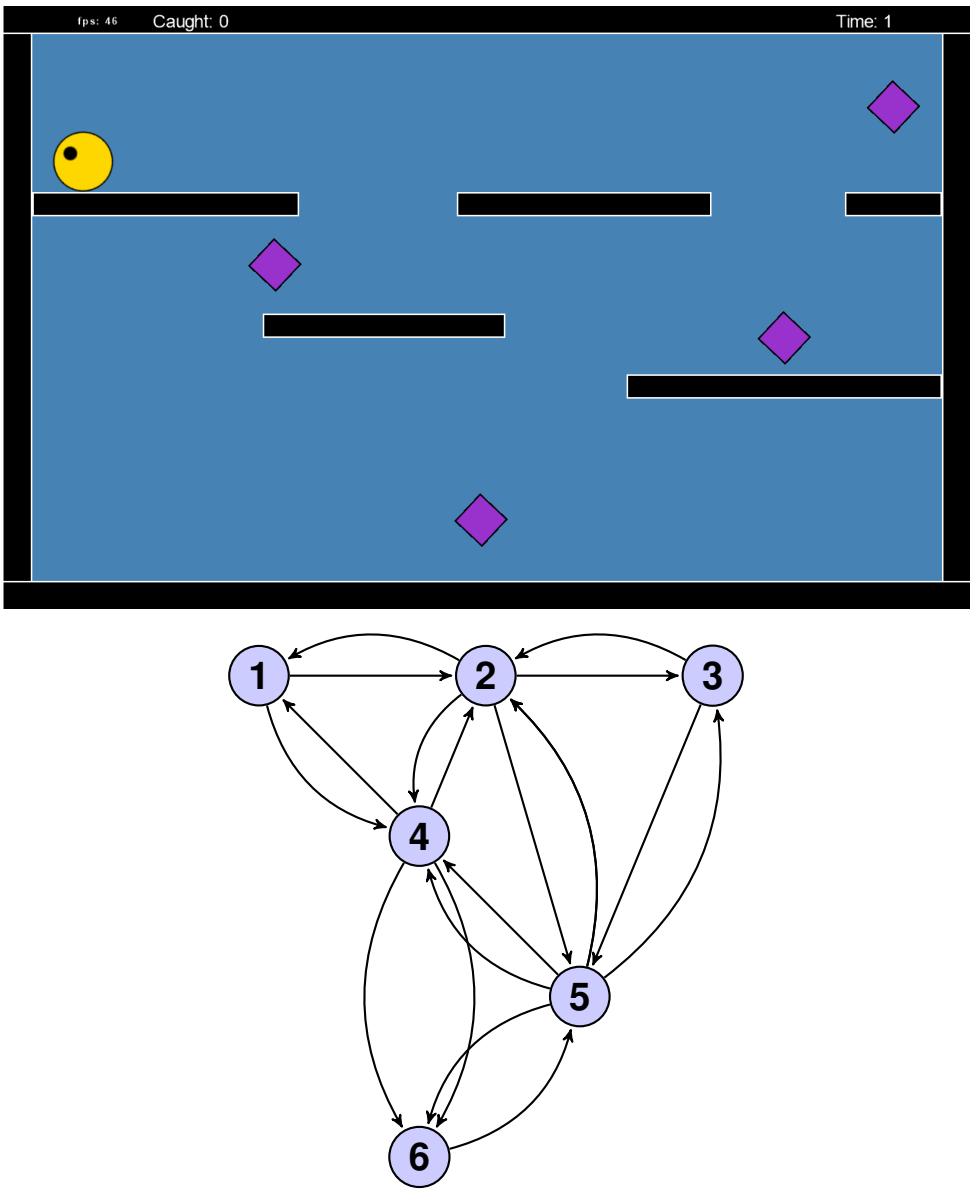


Figure 4.4: The navigation graph models the level that is shown above. The nodes represent the obstacles on the level and the ground. There is one edge whenever it is possible to go from one platform to the other. Notice the two edges from node 4 to node 6.

s

4.3 Moving to another platform

After deciding to where the character has to go, the agent must produce a set of actions that makes it reach that target platform. As it was stated before, this sub-problem is also modelled as a learning problem. The reason for choosing a learning approach for this problem has to do with the fact that the agent can only take into account the two platforms (the platform it is currently on and the one where it needs to go to) and ignore all other platforms on the game.

The major challenge on SP3 is to correctly control the speed of the character on the moment where it changes platforms, as failing to do so can result on the character landing outside the platform. In addition, if the character needs to jump to reach the intended platform, it needs to correctly predict the location where that jump action has to be performed. The location of the jump is tightly coupled with the speed that the character has, since higher speeds require earlier jumps.

To successfully solve SP3, the feature-vector has to capture the distance between the character and the jump-point the agent is targeting, the characteristics of the destination platform and the distance vector between the current platform and the destination platform. Moreover, the feature vector must also capture the character's speed. When moving to another platforms, the agent looks to the following features of the environment:

- Agent speed – integer value that ranges from -10 to 10 (uses the same 1/20 constant used when solving SP1);
- Distance to jump-point – integer value that indicates the distance in the xx axis between the character's current position and the jump-point position. This distance is also measured as the number of circle radii;
- Distance vector – two-dimensional vector that stores the difference in the xx and yy axis between both platforms. It also uses the same metric as the distance to the jump-point.
- Landing platform size – the portion of the destination platform that can be used for the character to land on it. It is also measured as the number of circle radii.
- The edge the agent is committed to – the edge of the graph the agent is committed to solve.

The visual representation of some of these features can be seen in figure 4.5

Whenever the agent reaches its goal, it stores the time-stamp of that moment on the edge the character traversed. The time-stamp is used to calculate how much time the agent used to move from one platform to the other. The reward function for a given state on this sub-problem takes into account:

- The fraction of the time the agent spent moving to the next platform. This time is measured from the first time the agent commits to an edge until it reaches the destination platform of that edge.
- The fraction of the time from the first time the agent committed to the edge ($initialEdgeTimeStamp$), to the time taken when the agent played was on that state ($stateTimeStamp$). The closer that fraction is to 1, the higher the reward the state gets, as it is an indication that the agent was closer to solving the sub-problem.

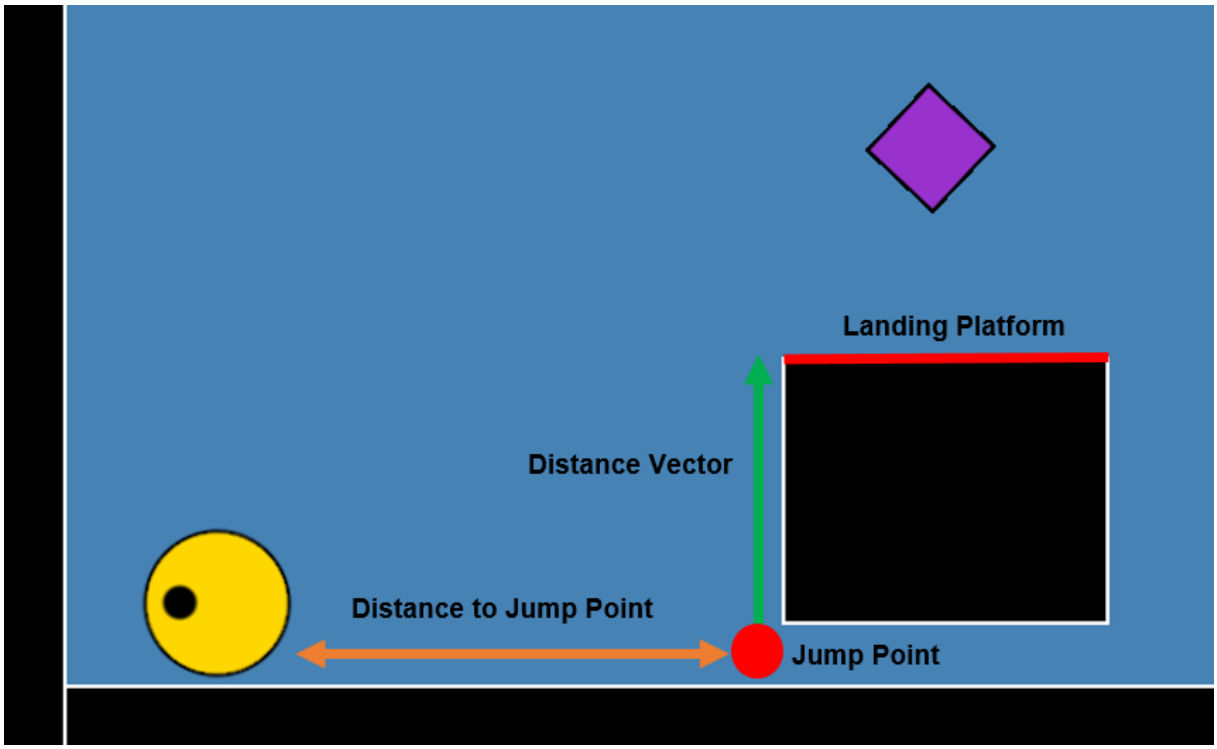


Figure 4.5: Features considered on SP3. The red dot is the jump-point the agent is committed to. The distance represented in orange is the distance to the jump-point value. The green vector is the distance vector. This vector also represents the edge of the graph the agent is going to traverse. Since the rightmost xx coordinate of the character's platform is the same as the leftmost xx coordinate of the destination platform, this vector xx coordinate is 0. The red line on is the size of the destination platform where it is possible for the character to land.

The reward function for a given state s is calculated using Equation (4.2), where $totalEdgeTime$ and $levelTimeLimit$ represent the total time the agent took to get to the destination platform and the level time limit, respectively.

$$\begin{aligned} reward(s) = 100 \times \left(1 - \frac{totalEdgeTime}{levelTimeLimit}\right) \\ \times \frac{stateTimeStamp - initialEdgeTimeStamp}{totalEdgeTime} \end{aligned} \tag{4.2}$$

Having the GF problem already divided in three sub-problems, each one with distinct characteristics it is important to understand how the agent deals with all of them. On the next chapter there are details about the architecture of the agent and the how the information flows from the moment the agent is prompted to perform an action to the moment that action is returned.

Chapter 5

Agent architecture

This chapter focuses on detailing the architecture of the agent developed on this work. The agent has five distinct modules, each one of them with its own set of functionalities. On section 5.1, there is a description of each one of those modules. Section 5.2 describes how the information flows through them and section 5.3 explains how the knowledge-base is structured and how the agent uses it.

5.1 Agent Modules

When the game engine asks for a new action, the agent has to get the latest updates from the game-environment and convert them into its own world representation. Using those updates, the agent generates a game-state object that represents the current state of the game. Finally, the agent queries its knowledge-base on how to proceed when facing that game-state. The architecture the agent was designed to encapsulate the behaviour of each one of those steps on a specific module. In addition, there is a module that orchestrates how the flow of information goes through the other modules. The five modules of the agent are:

- The World Interface – is responsible for getting the latest updates from the game-environment;
- The World Model – stores the representation of the current state of the world;
- The Platform Manager – is responsible for generating and storing all the information about the platforms of the level;
- The Learning Center – manages the connection between the agent and the knowledge-base. It is also responsible for updating the reward values on the end of a level;
- The Controller – orchestrates the whole flow of information, from the moment the agent is prompted for an action, until the moment it is returned;

Figure 5.1 presents a class diagram with all the five modules of the agent.

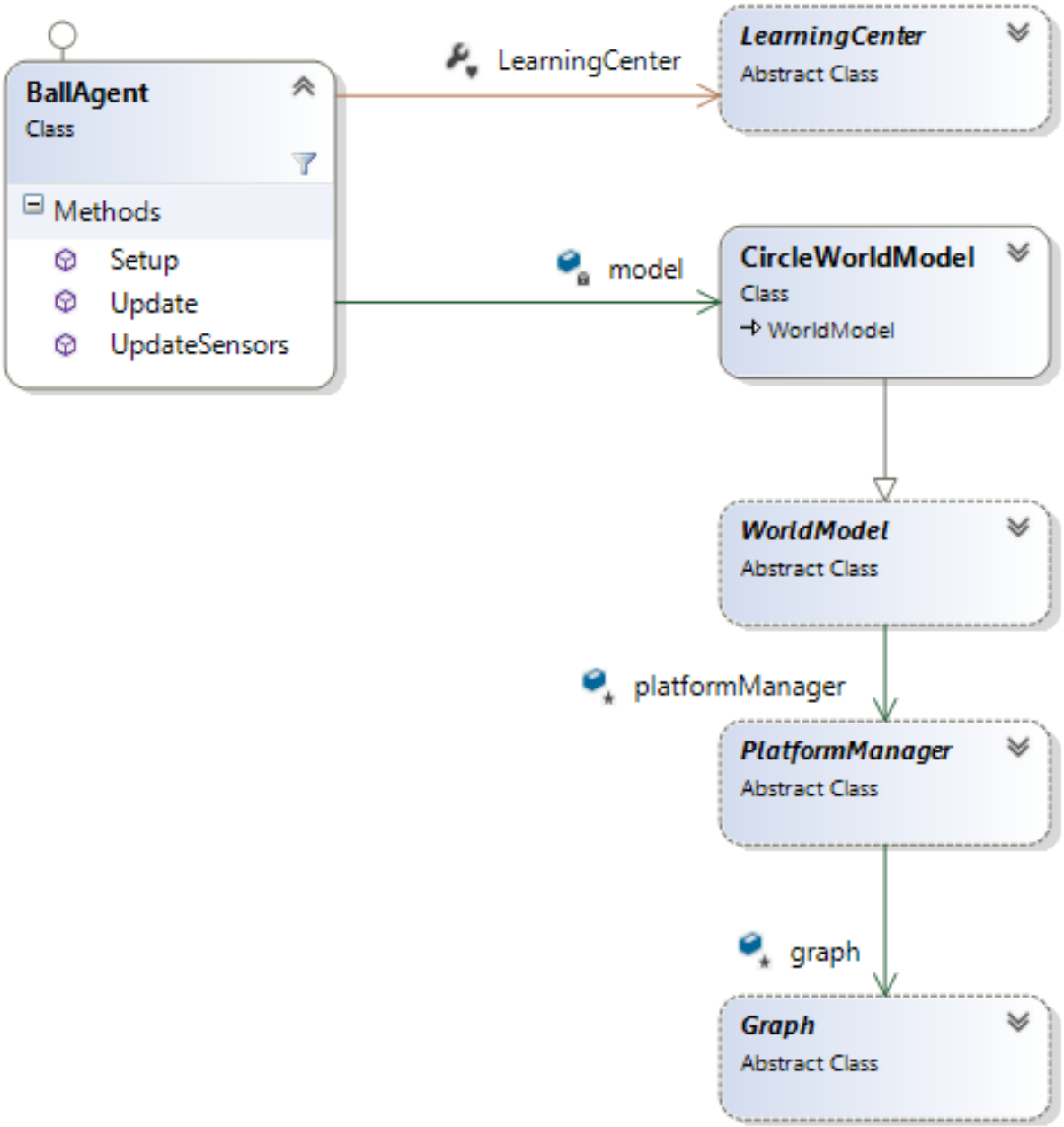


Figure 5.1: The agent class diagram. Image taken using the Visual Studio Code Diagram tool. The *BallAgent* class represents the Controller module. The three methods on this class are used by the World Interface to post the latest updates from the game environment. The *CircleWorldModel* is the implementation for the circle agent of the generic World Model

5.1.1 World Interface

The World Interface module (represented by the three methods of the *Ball Agent* class) makes the connection between the agent and the game engine. This component is already provided by the GF AI framework. The module contains the sensors that capture the latest updates from the game environment. The World Interface is called by the game engine in an initial set-up phase and whenever the agent needs to perform an action. The information captured by the sensors when the agent is prompted for an action are:

- The number of available diamonds on the level;
- The position of each one of the available diamonds;
- The position and speed of both characters;

Additionally, on the set-up phase, the sensors also capture

- The number of obstacles;
- The initial position of each one of the obstacles;
- The size (height and width) of the game environment;
- The time limit for the level.

The information gathered by this module is then passed to the World Model module, where it is converted into the agent's own world representation.

5.1.2 World Model

The World Model module is responsible for storing the agent representation of the game environment. The information that is stored here is either directly extracted from the World Interface sensors, such as the character's current position and radius, or calculated by a combination of those features. An example of a calculated feature is the number of diamonds that were already caught. To calculate that value, the World Model uses the initial number of diamonds, set on the initial set-up phase, and the current number of diamonds available, retrieved by the sensors when the agent is prompted for an action

The representation of the game environment stored in the World Model contains the following attributes:

- The character's position (2D Vector);
- The character's speed (2D Vector);
- The character radius (Integer number);
- The level time limit (Integer number – number of seconds);
- The number of diamonds that were caught (Integer number);
- The time elapsed since the start of the level (Float number – number of seconds);
- The position of the closest diamond (2D vector).

This module is also responsible for building the game-state objects that are then used by the agent to query its knowledge base. Since those objects need information about the platform the character is in (and sometimes about the platform the agent should move to) it needs to query the Platform Manager module.

5.1.3 Platform Manager

The Platform Manager module stores all the information about the platforms in the level. The module is responsible for generating all the platforms and the navigation graph (used to solve SP2) in an initial set-up phase. It is also the Platform Manager that is responsible for the assignment of the diamonds to the platforms.

A platform is generated whenever it is possible, for the circle, to go from the leftmost xx coordinate to the rightmost xx coordinate of an obstacle without colliding with any other obstacle. To do so, for each obstacle o in the game the Platform Manager creates a ”dummy” platform whose width is the same as the obstacle o and whose height is the diameter of the circle. This ”dummy” platform is positioned on top of obstacle o. Whenever this ”dummy” platform collides with another obstacle on the game environment, obstacle o has to be split into 2 distinct platforms $p1$ and $p2$. This process is repeated until there is no collision of ”dummy” platforms with other obstacles. Figure 5.2 shows how the ”dummy” platforms algorithm works. On that image, obstacle $O4$ has to be split into two platforms – $O4\_1$ and $O4\_2$ – whilst obstacle $O2$ hasn't.

The same ”dummy” platform technique is used to create the graph edges. A ”dummy” platform is created on the gap between the two platforms. If there are no collisions with an obstacle then it is possible to create an edge on the graph. The top of the ”dummy” platform is a 3 times the circle radius higher than the yy coordinate of the highest platform of the two to allow the circle to roll on top of that same platform. A depiction of the usage of the ”dummy” platforms to create the edges can be seen on Figure 5.3.

Whenever the ”dummy” platform has no collision with other obstacles, an edge between the two platforms can be created. However, for an edge to be created, the difference in the yy axis between the lower and the higher platform must less or equal to 8 circle radii (empiric constant that represent the maximum height that the circle can achieve with a jump action). The xx distance is not taken into account when creating an edge. This simplification can lead to the creation of some edges that cannot be traversed by the character, due to the large distance in the xx axis. Nevertheless, such scenarios are very rare on GF.

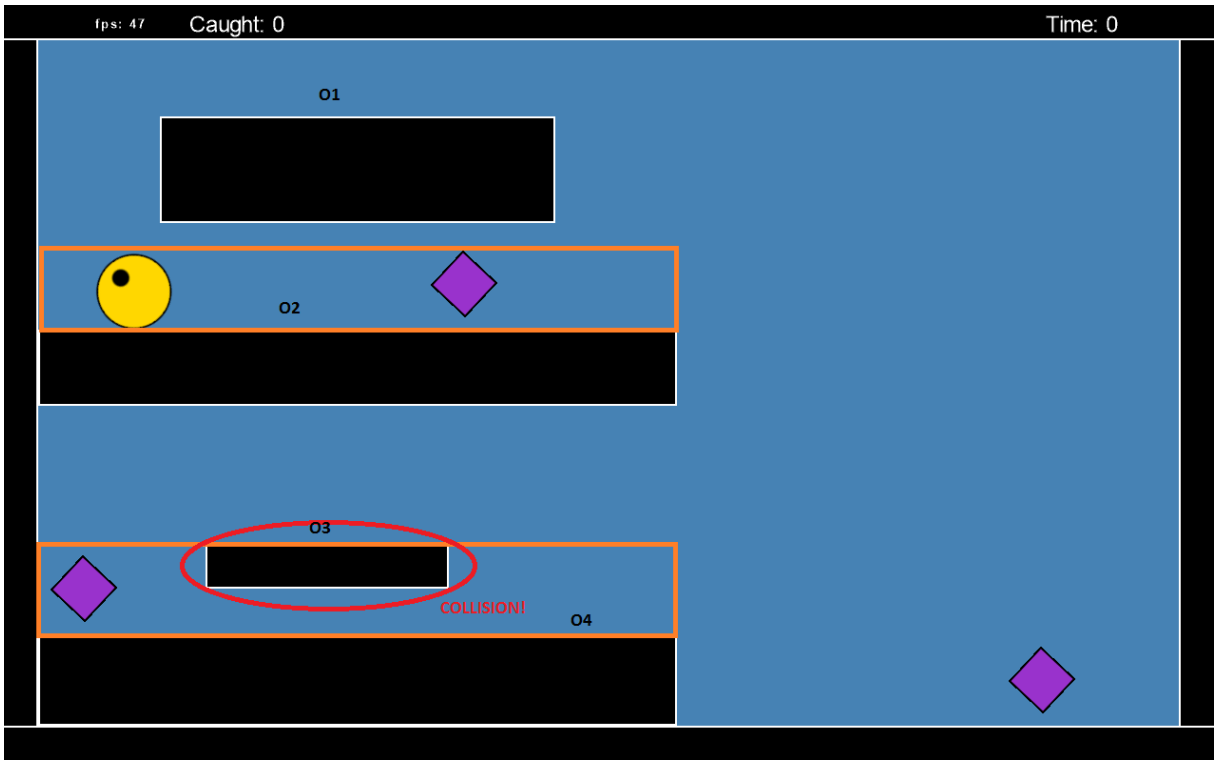
The Platform Manager is called by the World Model whenever it needs to discover in which platform the character is and when there is an update triggered by the World Interface. During the update, this module verifies, for each platform, if the number of available collectibles is the same and changes that value accordingly.

The Platform Manager stores the following information:

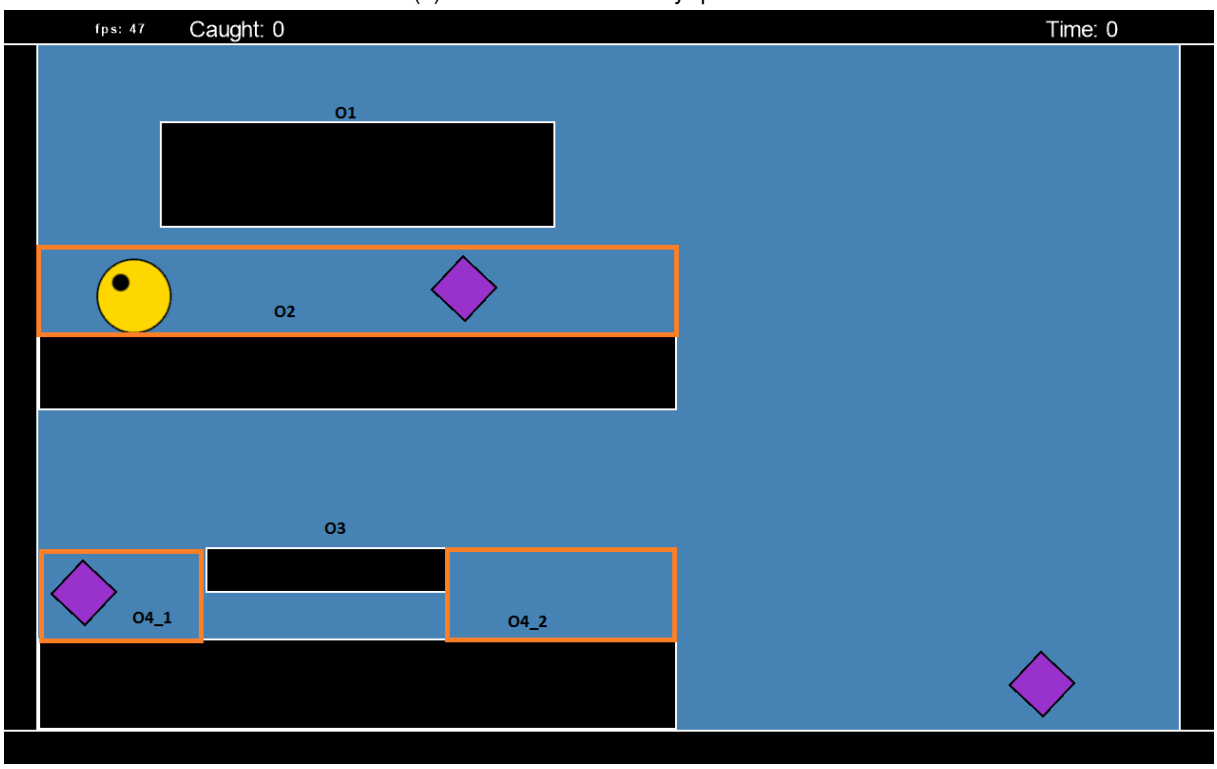
- The navigation Graph;
- The number of platforms that are on top of black obstacles;
- The number of platforms that are on top of coloured obstacles;
- For each one of the platforms:
  - Its position;

- The topmost and bottommost yy coordinates;
- The leftmost and rightmost xx coordinates
- The initial number of diamonds assigned to the platform;
- The current number of diamonds assigned to the platform;
- The position of those available diamonds.

Using the information stored in the Platform Manager, the World Model module has all the information it needs to generate an object that represents the current game state. The Controller module uses that object to query the Learning Centre for the best known move for that specific game-state.



(a) Calculation of ”dummy” platforms.



(b) After the split of the platforms.

Figure 5.2: Generation of platforms. The orange-lined rectangles represent the ”dummy” platforms. When these ”dummy” platforms collide with any obstacle on the level the obstacle has to be split into two distinct platforms.

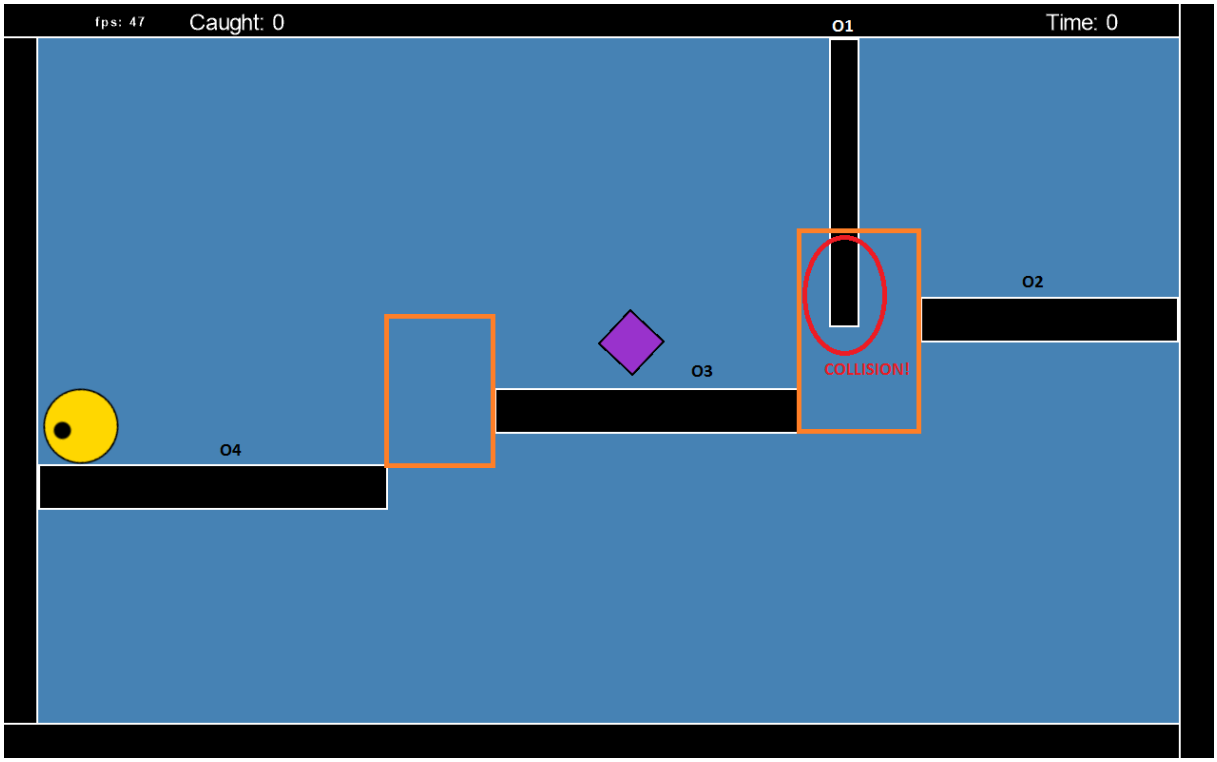


Figure 5.3: Generation of the navigation graph edges. The orange-lined rectangles represent the ”dummy” platforms. It is possible to create an edge between O4 and O3, however it is not possible to create an edge between O3 and O2. The ”dummy” platform that fills the gap between O3 and O2 collides with obstacle O1. Notice that the ”dummy” platforms go higher than the highest platform of the two.

5.1.4 Learning Centre

The Learning Centre is the module that is responsible for storing and updating the knowledge-base of the agent. In the set-up phase, this module loads the knowledge-base that is persistently stored into memory. The knowledge-base has a mapping between game-state identifiers and reward values for each one of the possible actions on that state.

When the game finishes, this module updates the knowledge-base and commits it to local storage. The details on how the knowledge is persisted are further discussed in section 5.3.

This module can either return the action that maximizes the reward for the queried state or an indication that such state was not found on the knowledge-base. The Controller module uses the Learning Centre output to decide which action the agent performs.

5.1.5 Controller

The Controller (*BallAgent* class in the class diagram) is responsible for making the final decision regarding the next action to be played. The controller queries the World Model and the Learning Centre to gather the information needed to make that decision. It is also the Controller that triggers the update of the knowledge-base when the level ends.

This piece of the architecture is responsible for orchestrating the flow of information from the moment the agent receives the perceptions to the moment it returns an action. The flow is described on the following section.

5.2 Agent decision flow

When the agent is prompted for its next action, it receives through the sensors located on the World Interface the newest information about the game environment. Once that information reaches the controller, it is passed to the World Model module to trigger the update of the agent's world representation. After that operation is concluded, the Controller queries the World Model for the identification of the game state.

The World Model, as soon as it finishes generating the game-state object, queries the Platform Manager for the platform where the agent is. If the agent is not in a platform, either because it jumped or because it fell from the platform it was in, then it stops all the processing and returns an Idle Action to the game engine (no forces are applied to the agent besides the ones generated by the physics engine of the game). However, if the agent is in a platform, then the World Model has to check whether that platform still has diamonds to be caught. If that's the case, then the agent is solving the sub-problem SP1 (Figure 5.4).

When solving SP1, the World Model uses only the data about that platform to generate the current game state identifier. In the case where no collectibles are available, there are two possible situations:

1. The agent hasn't decided to which platform it should move to;

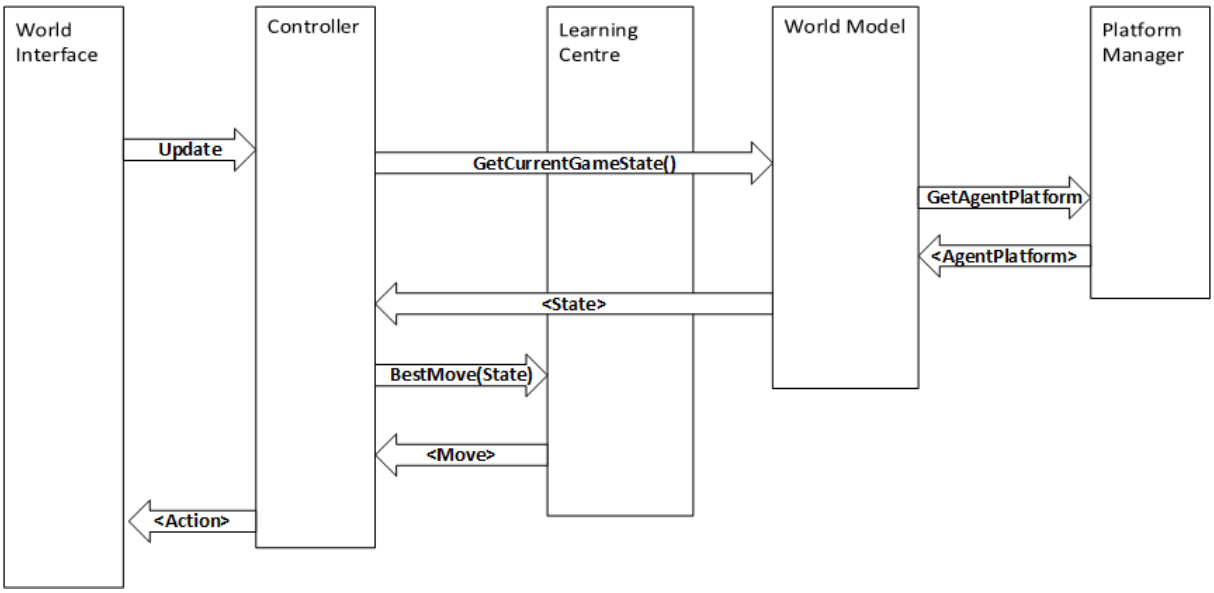


Figure 5.4: The agent decision flow on SP1. The arrows represent the direction of the flow of the information.

2. The agent has already decided to which platform it should move to;

On the first scenario, the agent firstly runs a DFS on the navigation graph that is stored in the Platform Manager. The search algorithm starts on the platform where the agent is. After the DFS finishes, the World Model has the set of edges that connect the character's current platform to the next platform of the computed path. Using the jump-point of each of one of those edges and the character's current position, the World Model calculates what is the jump-point that lies closer to the character. The agent will be committed to that jump-point. As soon as that jump-point is calculated, the agent uses the information of the agent's current platform, of the jump-point and the destination platform to generate the game-state identifier.

On second scenario, the DFS is also ran as a safeguard from the game environment non-determinism. However, if the first platform in the computed path is the same as the destination platform of the edge the agent is already committed to, then the agent maintains that commitment and skips the closest jump-point calculation step. The flow diagram with the DFS is depicted on Figure 5.5.

As soon as the World-Model returns the game-state identifier, the Controller queries the Learning Center for the best known move for that state. The Learning Center returns to the Controller either the movement to be played or an indication that the state was not found on the knowledge-base. If the state is not found, the Controller chooses a random action from the set of possible actions. On the other hand, if the state is found and a move is returned, then the Controller can make one of two decisions:

1. Play the action that was returned by the Learning Centre – This is the action that, according to the agent's present knowledge, maximizes the reward function on the agent's current game-state;

2. Play a random action, thus ignoring the decision made by the Learning Centre.

Each one of these decisions has a given probability of being taken by the Controller. For the first decision, that probability is called exploitation, whereas for second decision it is called exploration. The

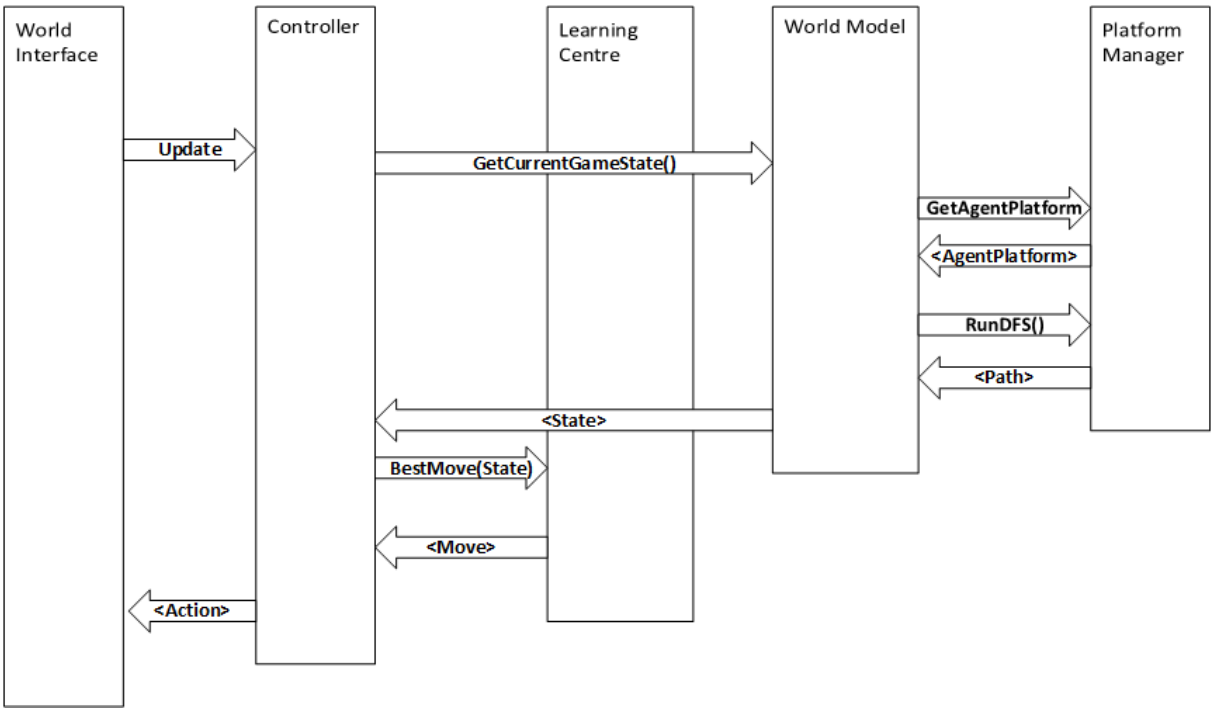


Figure 5.5: The agent decision flow on SP3. The arrows represent the direction of the flow of the information. Notice the 2nd call on the Platform Manager. It is this call that triggers the DFS

sum of the exploration with he exploitation probability is always equal to one. When the agent is training, it should favour exploration over exploitation so as to discover more unknown state. However, during the test runs, the agent should exploit the knowledge it acquired before. For these reasons the exploitation rate was set to 80% when playing in the test set (20 % of exploration) and 40% when the agent was training (60 % of exploration).

The action set used by the Controller are three of the Circle's possible moves:

- Roll to the left;
- Roll to the right;
- Jump.

In this work the morph movement was not taken into account. The reason for this exclusion was the fact that it is possible to achieve the same outcome using a combination of jump and roll actions. The non inclusion of the morph action reduced by 25% the number os possible pairs $< state, action >$ the agent has to learn.

There are some particularities on the random choice of actions, as the 3 actions do not have the same probability of being chosen. The jumping action takes much more time to terminate than any other action and it is not possible to reverse the outcome of that action while the agent is on the air. Hence, this action has to have a smaller probability of being performed randomly. The more unnecessary jumps the agent makes, the slower the learning becomes, because less movements are possible to be made during that level. To avoid frequent jumps, this agent only has a 5% probability of choosing the jump action when choosing a random action. Table 5.1 summarizes the probability of each action when the

Table 5.1: Actions' probability when the agents plays randomly. When the agent plays randomly but has recognized the state it is on, then these probabilities must be multiplied by the exploration ratio.

Action	Probability
Roll Left	0.475
Roll Right	0.475
Jump	0.05

agent is randomly playing.

After the Controller decides the action to be played, it returns it to the game engine. The next section of this chapter details how the knowledge is stored, loaded and updated by the agent.

5.3 Learning Process

All the learning components of the agent are centralized in the Learning Center module. The details on how the learning is implemented on the agent can be divided into three distinct topics:

- How the agent queries the knowledge-base;
- How the agent updates the knowledge-base;
- How the agent persists the knowledge-base.

Using the knowledge-base

As is has already been stated, the agent runs a set-up phase before making its first movement on the game. During that phase, the agent loads to memory its knowledge-base. This knowledge-base is split into two distinct structures: one containing knowledge related with SP1; other containing knowledge related with SP3. Both those structures are dictionaries that are indexed by the game state identifier. The game-state identifier is a string that contains the values for the features that are considered in each of the sub-problems. The game-state identifier for each one of the sub-problems is built on the following way (the parameters between square brackets are optional):

SP1 – /-Pos:distanceToLeftEdgeOfPlatform /+Pos:distanceToRightEdgeOfPlatform /Speed:characterHorizontalSpeed /Colls:numberAvailableDiamonds /ColVector:distanceVectorToClosestDiamond [/End:wall];

SP3 – /-Pos:distanceToLeftEdgeOfPlatform /+Pos:distanceToRightEdgeOfPlatform /Speed:characterHorizontalSpeed /VectorToJP:distanceToJumpPoint /JumpVector:distanceVectorBetweenPlatforms /LandingSize:platformLandingSize.

For each entry of the knowledge-base, there is a table that maps actions to rewards (figure 5.6). There are as many entries on the table as the number of actions the agent has already performed when it was on that specific state. When there is no reward for any action on a state, then that state is ignored. Whenever the Learning Center is queried for the best action for a specific state s, it first checks if s was

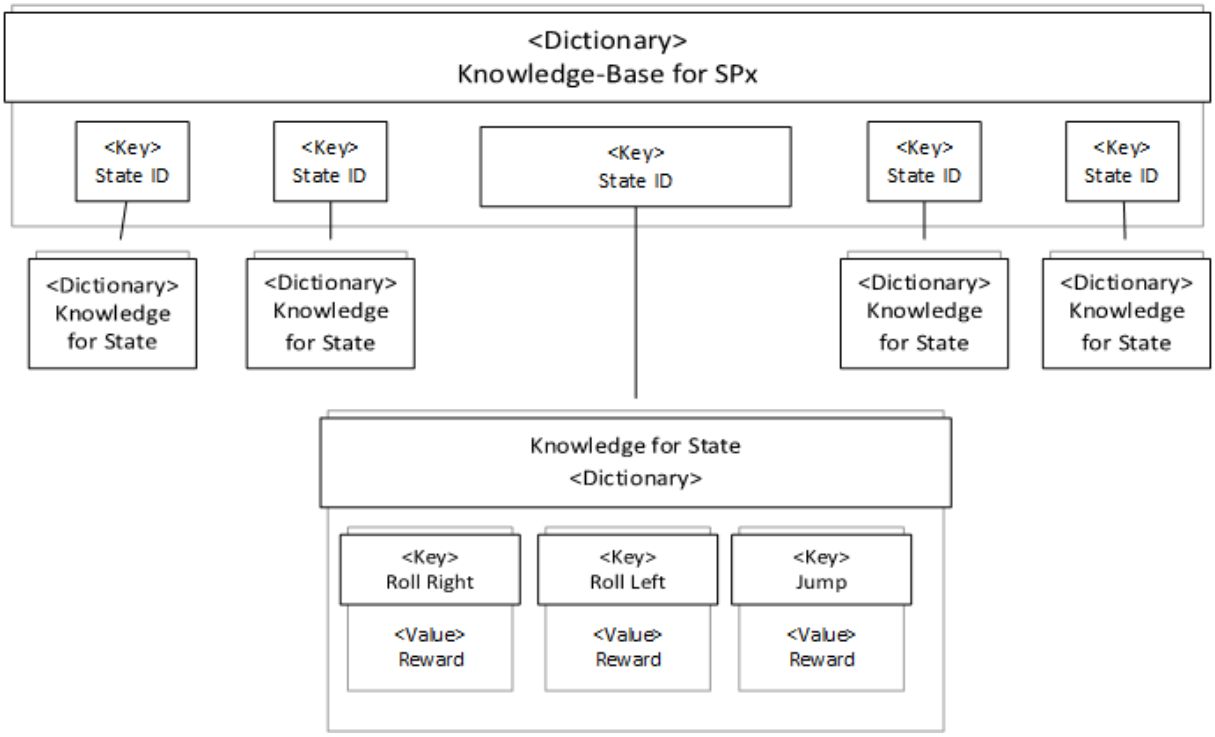


Figure 5.6: In memory representation of the knowledge-base. This structure is used for both SP1 and SP3 related knowledge, hence the SPx denomination. The knowledge-base is indexed by the game state identifier. For each state there is a table that maps actions to rewards.

already played and returns the action that maximizes the reward from the table of s. If s is not found, then the Learning Center returns a null object.

Updating the knowledge-base

The knowledge-base is updated every time a level is finished. The agent logs, in the Learning Center, all the actions it performs during the game and the state the agent is in. Those logs are already stored in two distinct sets, one containing the actions made when the agent was solving SP1 and other containing the actions performed when solving SP3. When the level (level n) ends, either because the agent solved it or the time limit was exceeded, the agent updates its knowledge for a given game-state s and an action a with the algorithm that follows.

if s *visited for the first time* **then**
  reward(s,a) =reward(s,a,n)
  reward (s,!a) = 0
else
  reward(s,a) = 0.95*reward(s,a,$n-1$) + 0.05*reward(s,a,n)
  reward(s,!a) = reward(s,a,$n-1$)
end

Notation: !a represents all possible actions different that are not action a. $n-1$ is the level that was played immediately before the one that has now finished. The reward values for level $n-1$ were the ones the agent loaded to memory when level n started. The constant 0.95 used to weight the known reward is the agent's learning rate.

The updated knowledge-base has to be persisted on storage in a way that the agent can load them

Table 5.2: Structure of a knowledge-base file. Each one of the two knowledge-base files has a table with this structure that stores the reward values for the possible actions of the agent. The 0 valued cells represent actions that were yet played on the state identifier by the StateId column.

StateId	Roll Left	Roll Right	Jump
A	100.2	123.55	0
B	0	0	22.9
C	56.7	22.5	10.2

when it is solving the next level.

Persisting the knowledge-base

The knowledge-base is persisted in storage as two files with identical structure. Each one of the files contains the fraction of the knowledge-base related to one of the two learning sub-problems. Those files have a very similar structure to one used in the in-memory representation of the knowledge-base. The files contain a table that, for each known state, has a column for the state identifier and one column to store the reward for each one of the possible agent actions. The reward column is filled with the values that are calculated during the update phase of the learning process, using the rewards that were already discussed in sections 4.1 and 4.3. Whenever there are actions that weren't yet played for a given state, the cells corresponding to its reward are filled with zero. Table 5.2 shows the structure of the table used to store the knowledge-base.

Currently the agent is storing this table as a $.csv$ file. A small excerpt of the SP1 knowledge-base file can be found in Appendix A (for readability purposes the file shown in the Appendix was exported on XML format).

This section concludes the architectural details of the solution. The next chapters will focus on how the agent was trained and on discussing the results it obtained in the GF AI Competition levels.

Chapter 6

Training

The training of an agent is a very important step when using learning algorithms. During such phase, the agent trains a vast set of possible situations in order to build a policy that can be used to solve the problems it is going to face during the test phase. In some lines of work, the agent starts the training phase with some basic knowledge, therefore using the training phase to tune that knowledge. However, in the solution proposed in this work, the agent stars the training without any prior knowledge. Hence, during the initial runs the agent always to plays random moves.

As has already been discussed, SP1 and SP3 uses different features to calculate a reward for a given state. Therefore, each one of those problems has a distinct training levels, which are designed to tackle that sub-problem's specific challenges. In each one of those training sets, both common and edge case scenarios are tested.

6.1 Training to solve one platform

To solve the problem of collecting all the diamonds on a platform, the agent has to learn which action it must produce in order to become closer to a diamond. In SP1, the more basic situation that the agent has to learn how to solve, is to solve platforms with only one diamond available. As soon as the agent learns how to move from its position towards the closest diamond, it can solve platforms with an arbitrary number of diamonds assigned to them (such as the situation depicted on Figure 6.1), provided that there is no risk of falling of the platform and no need to jump to catch a diamond.

Another basic scenario that the agent has to train is to catch a diamond that needs a jump action to be caught. To train such situation, there are training levels with several diamonds that are positioned at different heights. The variety of heights of the diamonds enables the agent to correctly capture the relation between the speed and the distance at which it should perform the jump action (figure 6.2).

If the agent has correctly trained the two above-mentioned scenarios, the agent is able to solve any platform, apart from the ones where there is the risk of the character falling off them. To deal with such scenarios, the computed policy has to take into account if a diamond is too close to the platform edge and if there is any obstacle to avoid the fall. If no obstacle is present then the character has to get to it

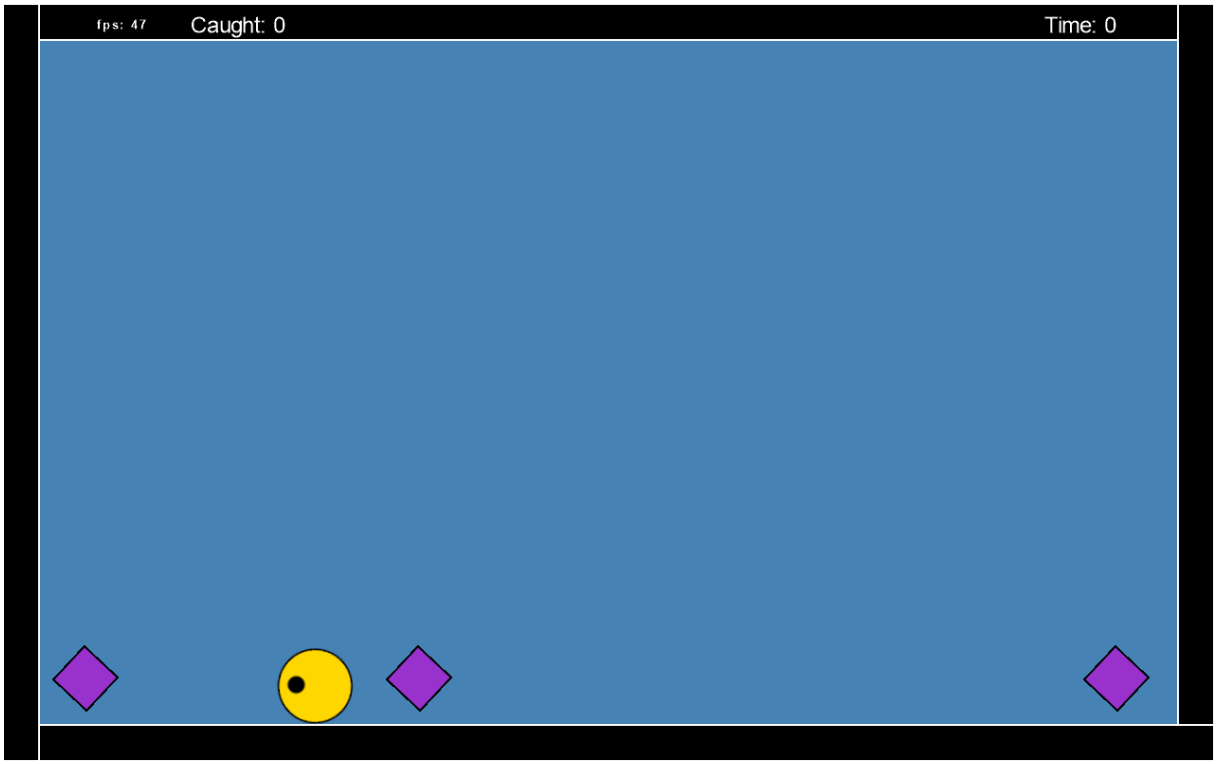


Figure 6.1: Basic roll action training level for SP1. Once the agent learns how to solve a level like this one, it is able to solve all levels where all the diamonds can be caught without any jumping action, are on the same platform and that platform covers all the game area width.

at a very slow speed. With that in mind, the training set has levels where one diamond is placed in each one of the edges of the platform. The platform id placed in a high location to prevent the character to get back to that same platform if it fell off it. Similarly to what was done in the ground platform situation, the scenario that feature diamonds that need a jump action to be collected is considered (figure 6.3).

In total, the training set for SP1 features thirteen distinct levels. The depiction of each one of the levels can be seen in Appendix B. To sum up, the levels of the training set considered the following situations:

- Rolling movements along a platform without the risk of falling;
- Catching diamonds that need a jump action but without the risk of falling;
- Rolling movement along a platform with the risk of falling;
- Catching diamonds that are on the edge of platform, with the risk of falling;
- Catching diamonds that are on the edge of platform and that impose a need to jump, with the risk of falling;

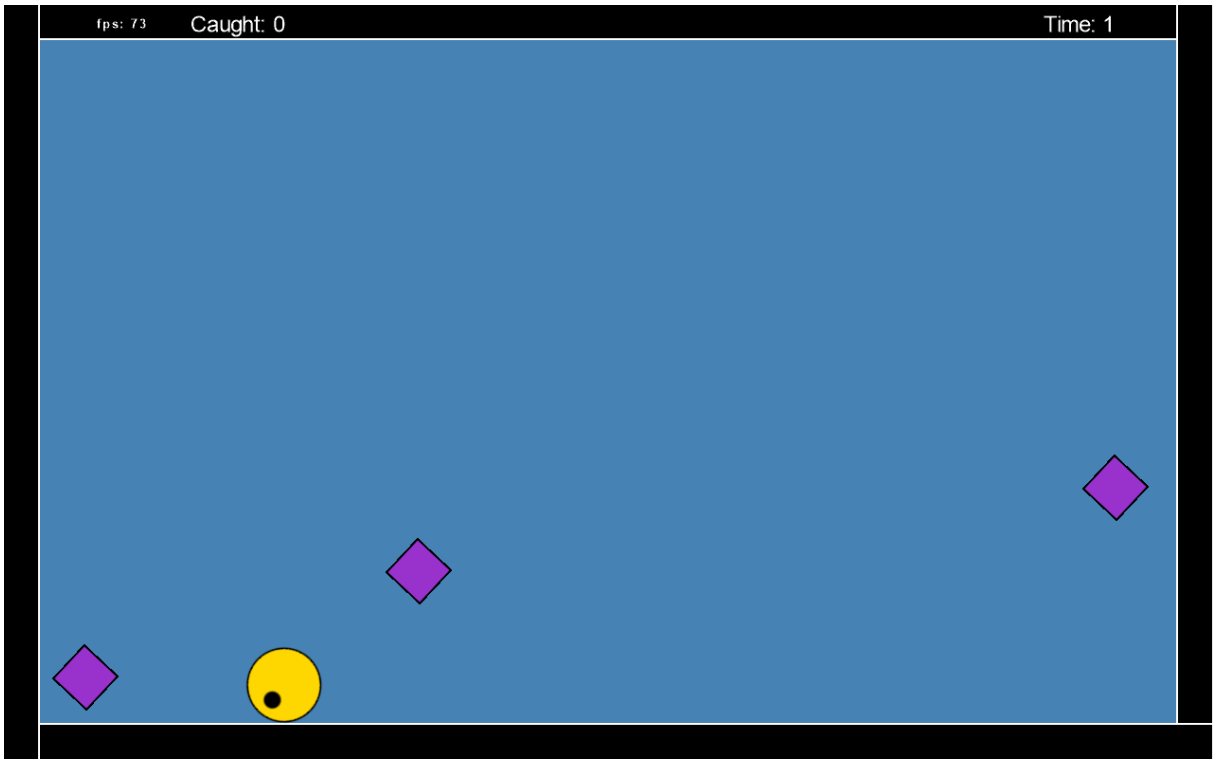


Figure 6.2: Basic jump action training level for SP1. Once the agent learns a policy for this level, it can solve the levels where all the diamonds are on the same platform and that platform spreads across all the game area width.

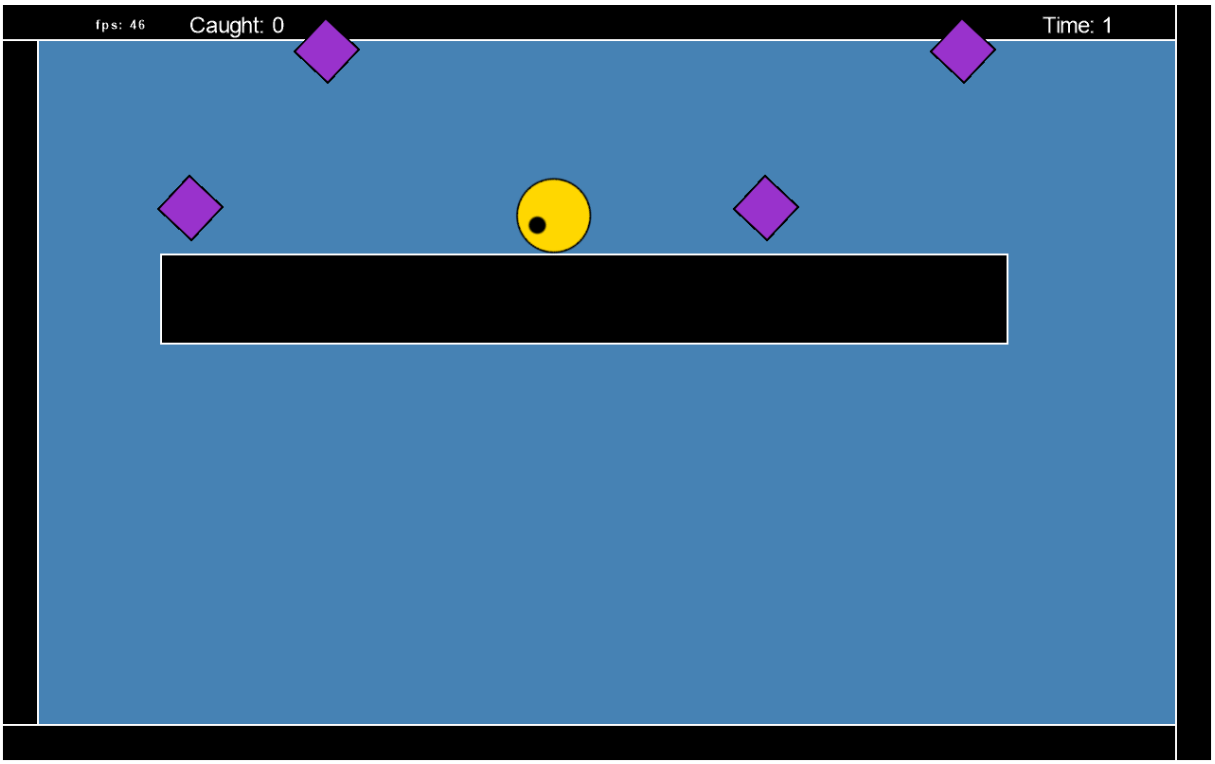


Figure 6.3: Training for a difficult situation in SP1. The training level requires the usage of both roll and jump actions to be solved. The agent must be careful when rolling towards the leftmost diamond as it can fall from the platform. It must also be careful when performing the jump action to catch the rightmost diamond.

6.2 Training to move from one platform to another

The training set for SP3 is also designed to capture both basic and complex situations that can appear when the agent is moving from one platform to another. The most basic situation occurs when the character has to roll until the end of the platform it is in order to fall to the platform below. After mastering this scenario, the agent is able to move from a higher platform to another independently of the height difference between the two (Figure 6.4).

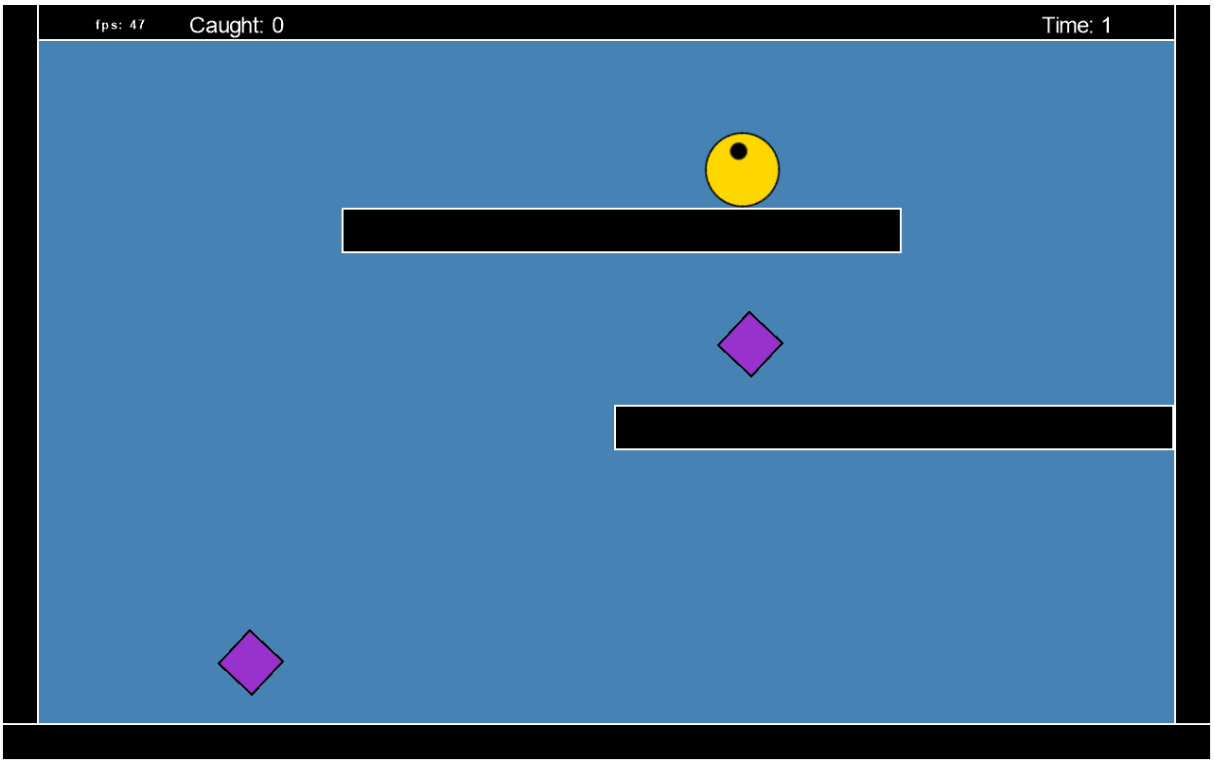
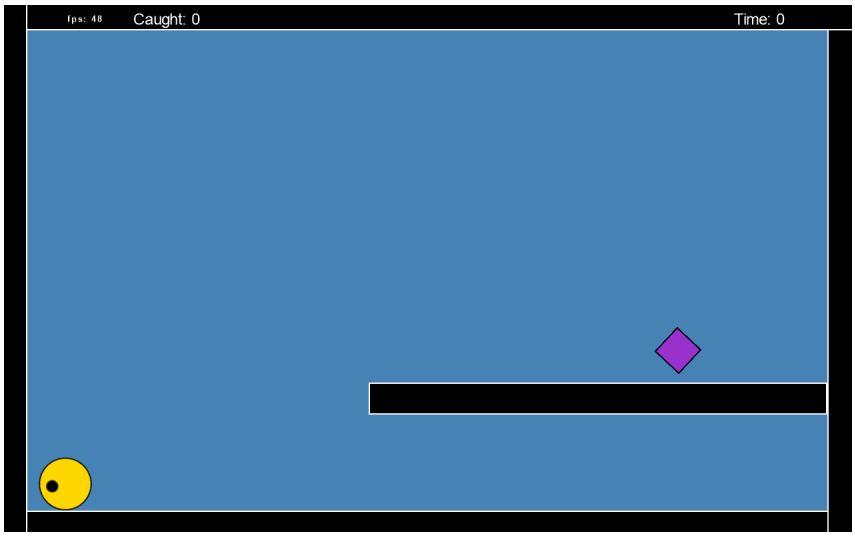


Figure 6.4: Basic roll action training level for SP3. Once the agent learns how to solve this level, it can solve every level where it only has to roll of the platform it is in to get to another platform.

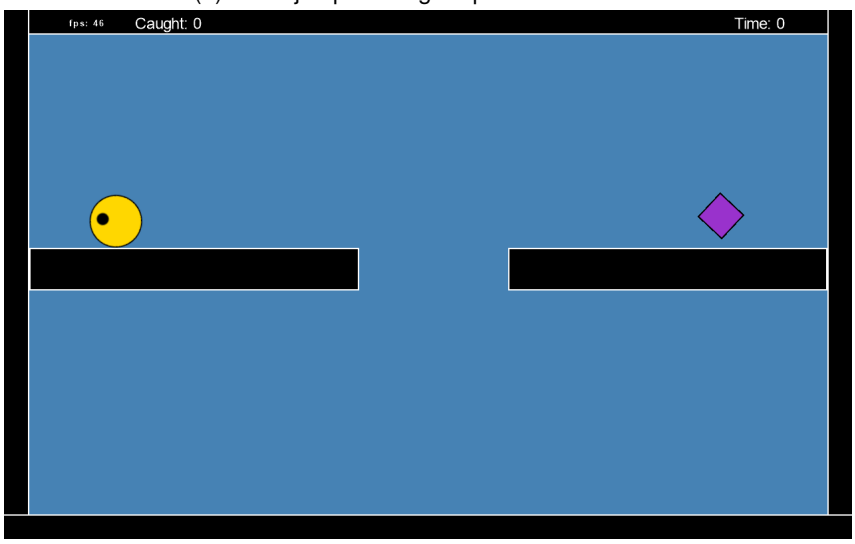
Another two basic and common situations in SP3 are the ones depicted on Figure 6.5. As soon as the agent can solve these two situations, the agent can solve all the possible scenarios of SP3:

- Go from a higher platform to a platform immediately below;
- Go from a lower platform to a platform immediately above;
- Go from a platform to a platform on the left;
- Go from a platform to a platform on the right;

Nevertheless, it is very important that the agent trains on situations that are either a combination of the four scenarios mentioned above or an edge case of one of them. One of such situations happens when the destination platform is small. In those cases, the agent has to learn to control its character's speed before jumping or rolling off the current platform so as to avoid failing to land on the intended platform.



(a) Basic jump to a higher platform scenario.



(b) Basic jump to a side platform scenario.

Figure 6.5: Basic jump training levels for SP3. Once the agent learns a policy to solve such scenarios, it is able to solve all the scenarios that can be imposed by SP3. However, the agent needs to be trained on situations that are a combination of both of these scenarios.

The hardest situation that can occur in SP3 is when there is a ”staircase”. The ”staircase” is a series of consecutive platforms (the ”steps”) that the character has to go through in order to reach the platform with the diamond. On such situation, the agent has not only to be able to jump to a higher platform, but also has to land with the correct horizontal speed in order to successfully perform the jump for the next platform. A level for training ”staircase” situations is depicted on Figure 6.6.

The training set for SP3 contains ten distinct levels. The depiction of each one of the levels is available in Appendix C. To sum up, the levels of the training set contained the following situations:

- Basic roll over to the platform below;
- Roll over to the platform below but with the need of controlling speed;
- Basic jump to a side platform (very small height variation);

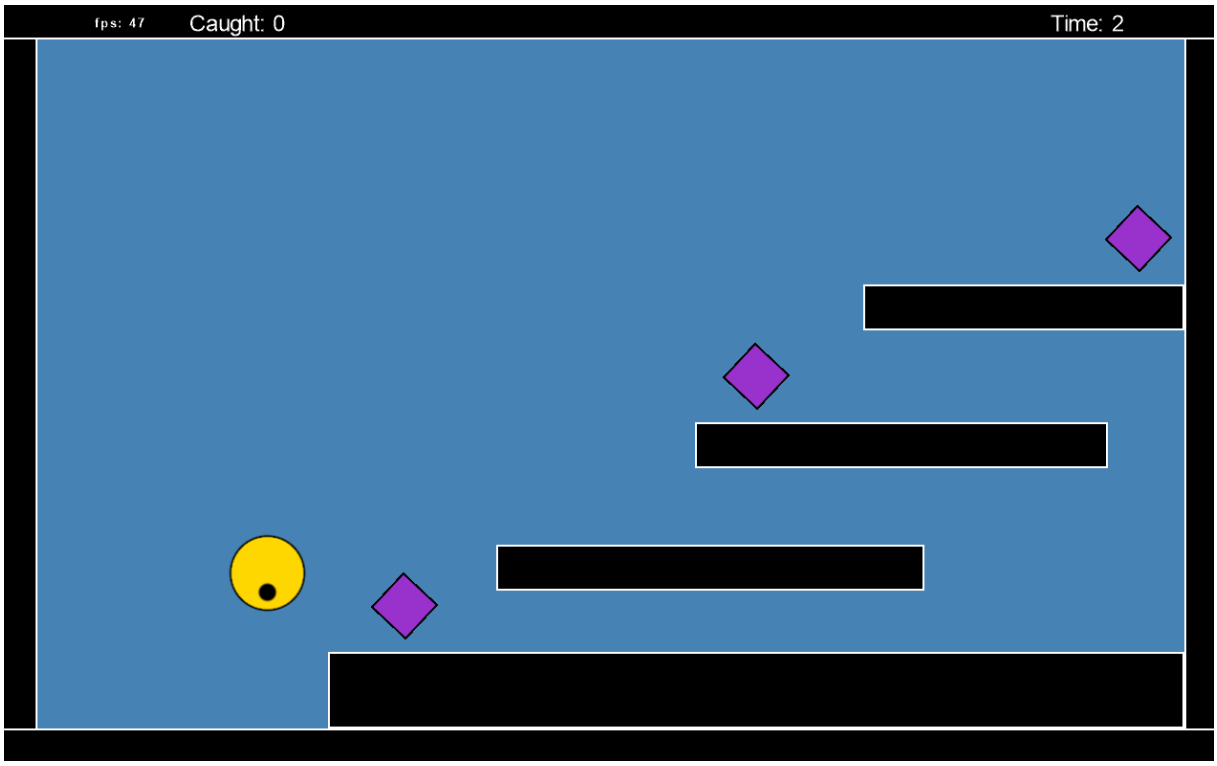


Figure 6.6: The ”staircase” scenario in SP3. The agent has to arrive at the destination platform with the correct speed to be able to jump to next ”step”.

- Basic jump to a higher platform;
- Jump to a side platform above the character’s platform;
- Jump to a side platform below the character’s platform;
- ”Staircase” situations (both climbing up and down the ”staircase”);

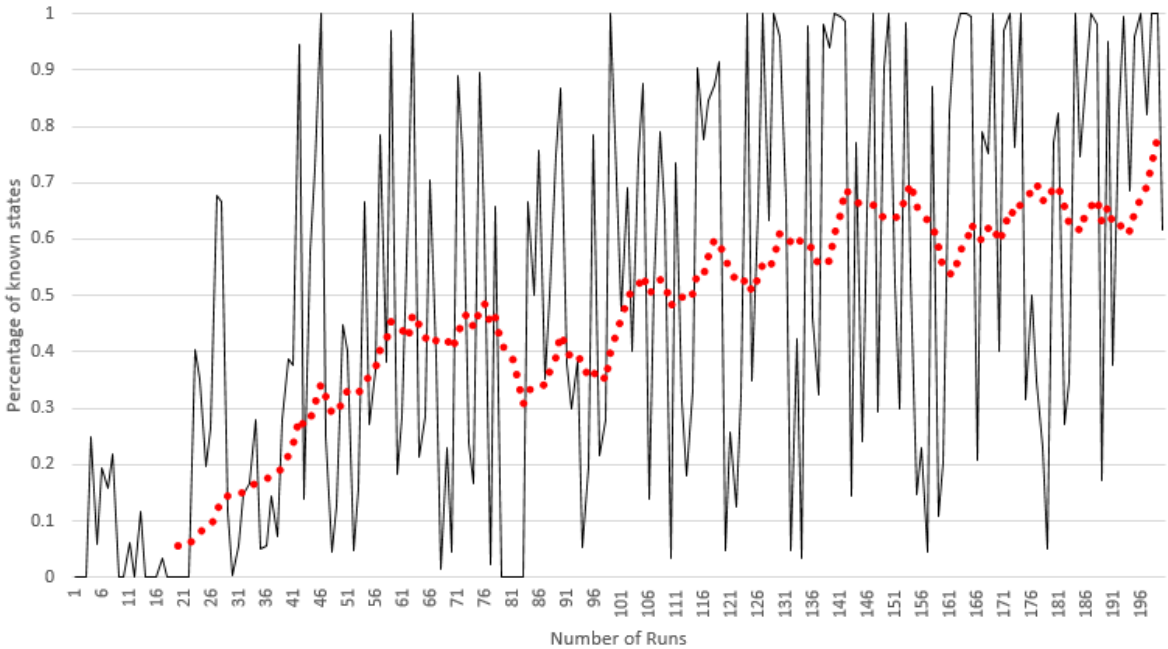


Figure 6.7: Percentage of recognized states for the first 200 runs of the 1st phase of training. This phase of training only comprises levels that are designed to train for SP1. The dotted red line is the moving average of the last 20 runs.

6.3 Training process

The agent had a three phased process of training. On the first phase, the agent had a total of 5000 runs on the SP1 training levels. Then, the second phase of the training consisted on another 5000 runs on the SP3 training set. Finally, the agent run continuously on a third training set containing both SP1 and SP3 training levels together with the 2014 Circle AI Competition public levels. The agent ran 9871 levels on this third phase.

In all training phases, the exploitation ratio was kept at 40% to bias the decision of the agent towards action whose outcome was not known yet.

To analyse the impact of the training on the agent, the percentage of states the were already on the knowledge-base was measured on each run. The results of those measurements for the first 200 levels of each phase can be seen in the three graphs depicted on Figures 6.7, 6.8 and 6.9.

As it can be seen from the graph on 6.7, after approximately 115 runs the agent starts to recognize, on average, more than half of the states it goes through. As it can be easily seen the average number of states recognized (dotted red line) increases with the number of runs played.

For SP3 (figure 6.8) the learning process is slower than on SP1. This is caused by the bigger number of possible situations that the agent can face, as it must consider both platforms characteristics. In this scenario, only after the first 160 runs the agent starts to recognize more than 20% of the states it visited. Nevertheless, the agent increases roughly 10% the percentage of recognized states in those first 200 runs.

When facing the whole training set, the agent recognizes between 60% to 80% of the states. Since the agent has already ran 10000 training runs, it starts this phase of training with a large number of

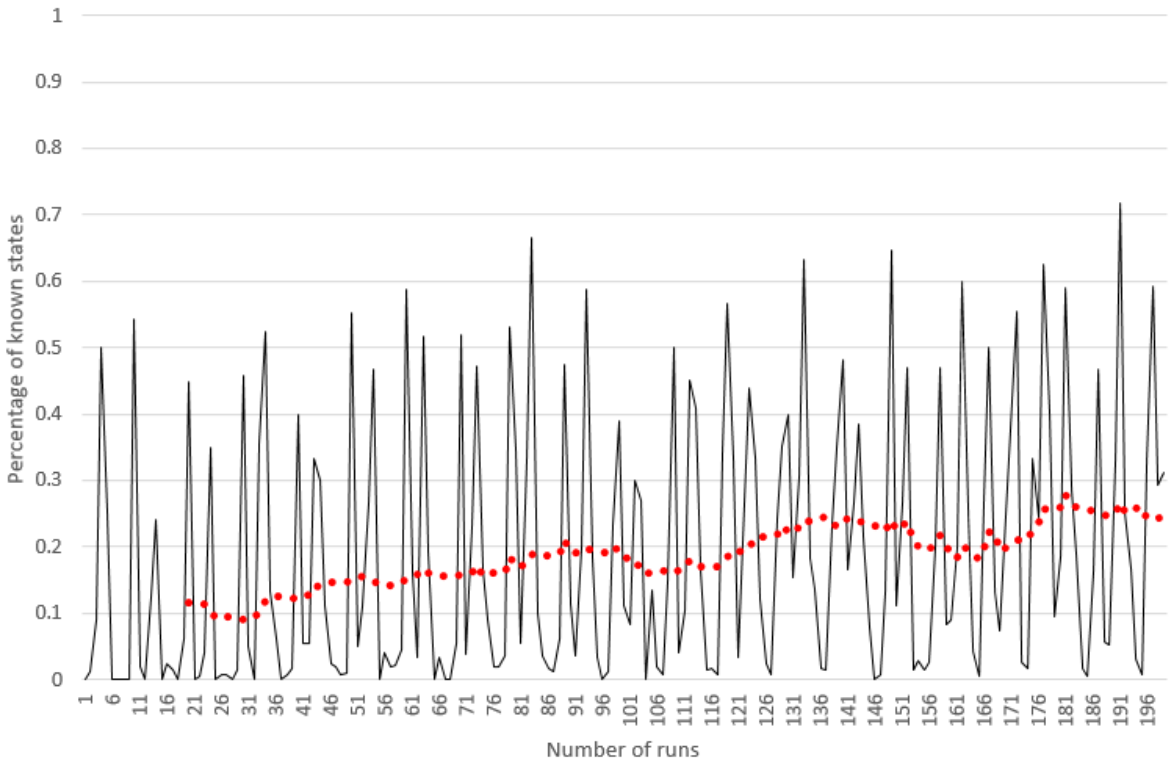


Figure 6.8: Ratio for the phase 2 of training (SP3). This phase of training only comprises levels that are designed to train for SP3. The dotted red line represent the moving average for the last 20 runs.

states on the knowledge-base. As a consequence, it is possible to notice that there are very few runs where the agent recognized less than 20% of the states. The moving average (red dotted line) fluctuates between 60% and 80% of recognized states. Finally, on the graph depicted on Figure 6.10 it is possible to see the percentage of recognized states on the last 200 runs of the training process. On this phase of training, the agent recognizes almost all states it goes through.

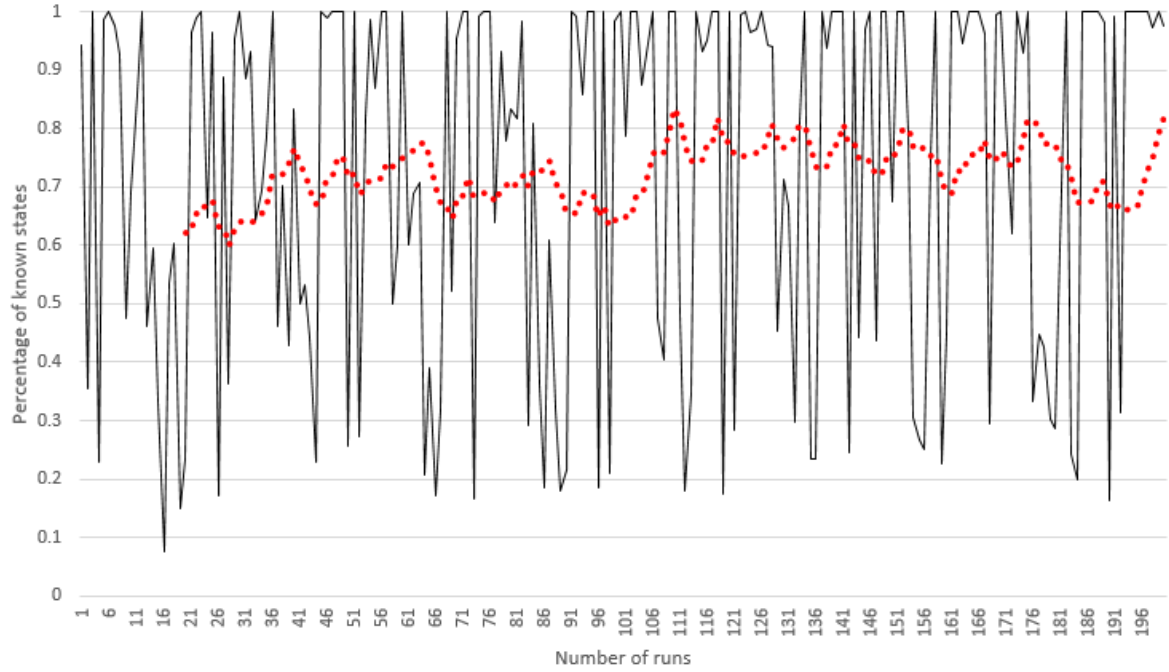


Figure 6.9: Percentage of recognized states for the first 200 runs of the 3rd phase of training. This phase of training comprises all the levels specially designed for the training process and the 5 public levels of the 2014 GF AI Competition. The dotted red line is the moving average of the last 20 runs.

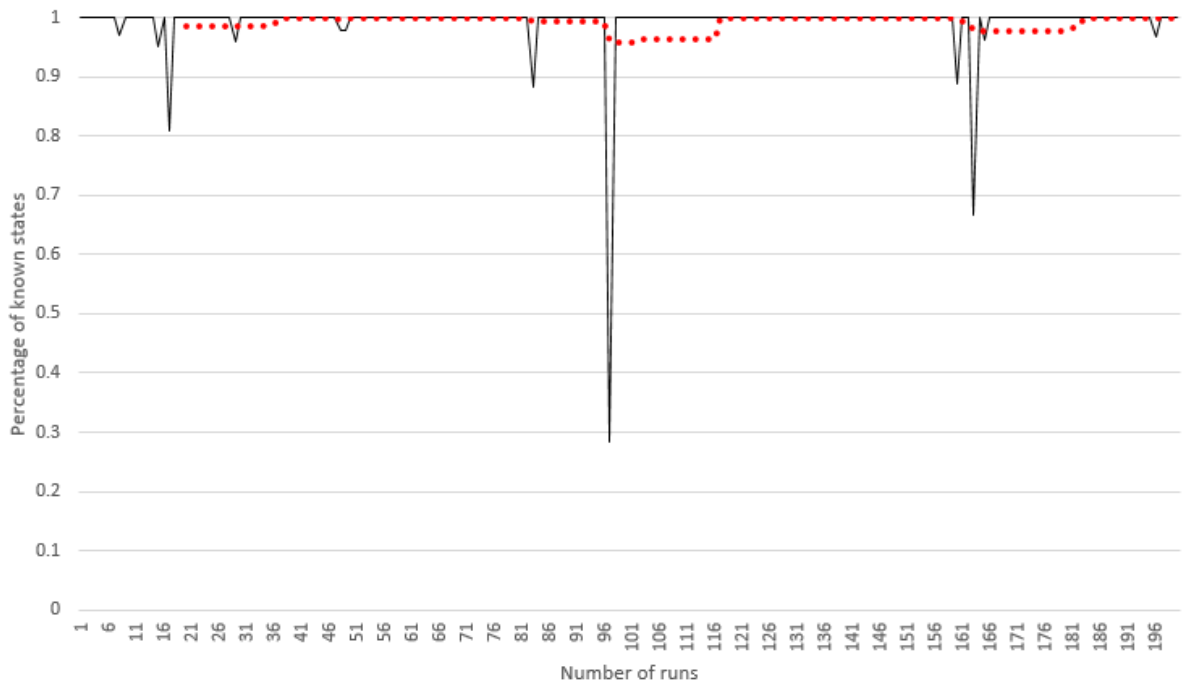


Figure 6.10: Percentage of recognized states for the last 200 runs of the training process. The agent recognizes all the states it goes thorough in almost all the runs. The dotted red line is the moving average of the last 20 runs.

Chapter 7

Results and discussion

7.1 Evaluation Process and Results

The agent was tested on the level set of the 2014's edition of the GF AI Competition. The tests were run in an Intel i7 processor at 2.4 Ghz and 16 GB of RAM. Windows 8.1 64-bit edition was used.

The results obtained by the agent (referred to as Agent α in this chapter) were compared with the results of both participants of the competition, CIBot and KUAS-IS Lab. Tables 7.1 and 7.2 present, respectively, the results of CIBot's and KUAS-IS Lab's Circle agents on the competition whilst table 7.3 presents the results for Agent α. To improve readability, whenever an agent was the best of the three on a particular level, that level's row on the corresponding agent's table is written in bold. Agent α was ran with the same rules and time constrains as if it was also in competition. In the competition the agents ran 10 times on the same level in order to mitigate the chance factor. In total, Agent α run 100 runs. The update of knowledge values was disabled to avoid learning between successive runs on the same level. The last 5 levels were completely new to Agent α. On the results' tables, the column Runs Completed shows the number of runs in which the agent solved the level and its value ranges from 0 to 10. The field Avg. Coll. represents the average number of diamonds caught by the agent in that level over the 10 runs. Between brackets is the number of diamonds of the level. Avg. Time is the average time the agent took to solve the level. The time limit for the level is shown between brackets. Finally the Score column is calculated by averaging the score of each run. The score of a run is obtained using Equation (7.1) which is the same that was used on the GF AI Competition. On that equation $V_{Completed}$ and $V_{Collect}$ are the bonuses for completing the level and catching a diamond respectively. In our tests, to mimic what was done in the competition, those values were set to 1000 for $V_{Completed}$ and 100 for $V_{Collect}$. $agentTime$ is the time the agent took to solve the level, $maxTime$ is the time limit for the level being played and $N_{Collect}$ is the number of diamonds caught.

$$\begin{aligned} ScoreRun = V_{Completed} \times \frac{maxTime - agentTime}{maxTime} \\ + (V_{Collect} \times N_{Collect}) \end{aligned} \tag{7.1}$$

Table 7.1: Results of CIBot circle agent

#Level	Runs Completed	Avg. Coll.	Avg. Time	Score
1	10	2 (2)	12.67 (20)	567
2	**10**	**3 (3)**	**19.89 (45)**	**858**
3	**10**	**3 (3)**	**14.84 (60)**	**1053**
4	0	1.2 (4)	80 (80)	120
5	0	1 (2)	70 (70)	100
6	0	1 (2)	40 (40)	100
7	**10**	**3 (3)**	**26.19** (60)	**864**
8	0	0 (3)	40 (40)	0
9	**10**	**3 (3)**	**50 (80)**	**675**
10	0	0 (3)	100 (100)	0
Totals	5 (avg)	1.72 (avg)	45.25 (avg)	4337 (sum)

Table 7.2: Results of KUAS-IS Lab circle agent

#Level	Runs Completed	Avg. Coll.	Avg. Time	Score
1	**10**	**2 (2)**	**5.81 (20)**	**910**
2	0	2 (3)	45 (45)	200
3	0	0 (3)	60 (60)	0
4	0	1 (4)	80 (80)	100
5	0	0 (2)	70 (70)	0
6	0	0 (2)	40 (40)	0
7	10	0 (3)	60 (60)	0
8	0	0 (3)	40 (40)	0
9	0	0 (3)	80 (80)	0
10	0	0 (3)	100 (100)	0
Totals	1 (avg)	0.5 (avg)	58.081 (avg)	1210 (sum)

Table 7.3: Results of Agent α

#Level	Runs Completed	Avg. Coll.	Avg. Time	Score
1	8	1.5(2)	13.00(20)	495
2	0	1.6(3)	45.00(45)	160
3	0	1.0(3)	60.00(60)	100
4	**0**	**2.0(4)**	**80.00(80)**	**200**
5	**2**	**1.2(2)**	**65.65(70)**	**182**
6	**6**	**1.4(2)**	**22.82**	**574**
7	1	0.8(3)	56.51	138
8	**3**	**2.3(3)**	**38.90**	**257**
9	2	2.2(3)	75.62	275
10	**0**	**0.6(3)**	**100.00**	**60**
Totals	2.2 (avg)	1.46 (avg)	61.61 (avg)	2441 (sum)

7.2 Discussion

As can be seen from the result's tables, Agent α would have placed second on the competition, scoring roughly half the points of the winner CIBot and approximately twice the points of the competition's runner up KUAS-IS Lab. The huge difference in the final score between the Agent α and CIBot's agent is mainly due to the fact that the reward for finishing a level is much bigger than the reward given for collecting diamonds. For example, KUAS-IS Lab scored more points only by solving the ten runs of level one

than Agent α scored for its combined performance on levels 4,5,7,9 and 10, where KUAS-IS Lab never managed to catch any diamond.

If the competition results were calculated using the number of levels where the agent had a better score, than Agent α would have won, with CIBot coming in second place:

Table 7.4: Classification of the agent by the number of levels won

Agent	#Levels Won	Levels Won
Agent α	5	4, 5, 6, 8 and 10
CIBot	4	2, 3, 7 and 9
KUAS-IS Lab	1	1

As this work's goal is to overcome the over-specialization on the public levels of the GF Competition it is important to analyse the performance of the agents on the public and unknown levels separately. In this analysis, only the average percentage of diamonds caught on each level is going to be presented.

7.2.1 Public Levels Analysis

The public level set comprises the first five levels of the competition. For this set of levels, the average percentage of diamonds caught is presented on Table 7.5. From the table it can clearly be seen that CIBot is the best agent in this set, collecting, on average, more that 3/4 of the diamonds on the levels. Agent α manages only an average of 54% diamonds per level, however it is the agent that has the highest minimum of the three agents presented.

Table 7.5: Average percentage of diamonds caught, per agent, in the public levels.

#Level	Agent α	CIBot	KUAS-IS Lab
1	75%	100%	100%
2	53%	100%	66%
3	33%	100%	0%
4	50%	30%	25%
5	60%	50%	0%
Average	54%	76%	38%

7.2.2 Private Levels Analysis

The private level set comprises the last five levels of the competition. The average percentage of diamonds caught on this level set is presented on table 7.6. From those results, it is possible to verify that Agent α is the best agent on the private levels, followed closely by CIBot. KUAS-IS Lab never caught any diamond on this level set.

7.2.3 Public Levels vs Private Levels comparison

The results show that Agent α is the only agent that has practically the same performance on both private and public level sets, having only an 1% difference in the average number of diamonds caught between both sets. CIBot and KUAS-IS Lab, on the other hand, are clearly over-specialized to the public

Table 7.6: Average percentage of diamonds caught, per agent, in the private levels.

#Level	Agent α	CIBot	KUAS-IS Lab
6	70%	50%	0%
7	27%	100%	0%
8	76%	0%	0%
9	73%	100%	0%
10	20%	0%	0%
Average	53%	50%	0%

levels. The difference of performance between the level sets are of 26% for CIBot and 38% for KUAS-IS Lab. Even tough Agent α is not the best in the competition, it clearly didn't become over-specialized in any set of levels. Recapping this work's main goals we have the following status:

- **Perform better than the winner of the IEEE CIG 2014 GF AI Competition,** CIBot **on the private levels of that competition;**
  - **Partially Accomplished**. Agent α had a better average percentage of diamonds caught and had a better score in three of the five levels of the private level set. However that level set's final score was worse than CIBot's (1639 points from CIBot against 1304 points from Agent α)
- **Collect an average of more than 50% of the diamonds and have a difference between the percentage of diamonds caught in the private and on public levels sets smaller than 10%.**
  - **Accomplished**. Agent α had a difference of 1% of diamonds caught between both level sets and caught an average of more than 50% of diamonds caught.

Even tough this work's goals were not completely accomplished, the final result is an agent that is capable of avoiding the specialization on a particular set of levels. This is a satisfying result that indicates that if some of the limitations of Agent α are overcome, the proposed approach can be used on an agent that, not only is capable of achieving a consistent performance on any level of GF, but also to challenge other AI Systems for the first place on future GF AI Competitions. Some of the limitations of the agent are presented in the following section.

7.3 Limitations

During the evaluation phase there were some limitations that were found on the agent. Out of those limitations, there is one that is due to the specificities of GF while the others are typical limitations of reinforcement learning problems.

7.3.1 Jumping problems

When the agent faces levels that have small platforms to which it needs to jump to, most of the times it jumps over the platform instead of landing on it. However, when the agent manages to land on the

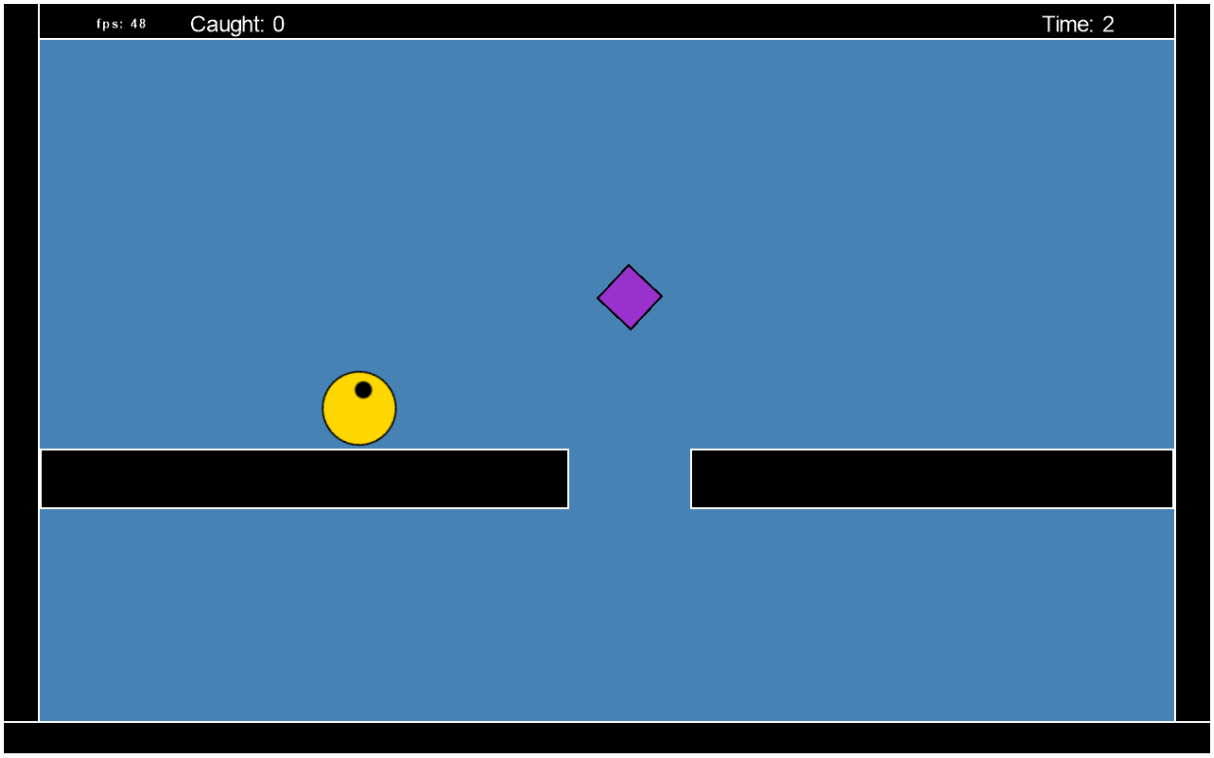


Figure 7.1: Diamond assignment limitation. The diamond is assigned to the ground platform but is impossible to catch it from there.

intended platform, it has the correct speed and rarely slips out of the platform. A possible explanation can be the lack of training on such situations, either because there was simply not enough training runs for the agent to learn how to tackle such situations or because levels didn't capture well these cases. Another problem is the fact that the agent is unable to correctly predict the movements needed to catch a diamond that has to be collected while the character is jumping from one platform to another. This problem is created due to the fact that the graph building algorithm assigns each diamond to the platform that is immediately below it. However, when the diamond is in a situation such as the one depicted in Figure 7.1, it will be assigned to a platform that is very far from it. A possible workaround would be to assign the diamond to one of the top platforms. However, that solution would require several changes to way the agent processes a single platform, as for this type of diamonds it would be impossible to just roll towards it. A more elegant solution, is to allow the diamonds to be assigned to the edges of the graph. A diamond can be assigned to more than one edge, as it can be possible to collect when moving from one platform to the other and the other way around. Moreover, the edges of the graph will also need to have a weight that represents the number of diamonds that are assigned to them. The DFS will have to find a path that goes through all the platforms and all the edges with diamonds.

7.3.2 Finding the next state

Another limitation in this solution is the fact that the agent sometimes makes an action but the game state that is generated after it is equal to the initial state. This creates confusion on the reinforcement learning algorithm, because for the same pair $< state, action >$ two different outcomes can occur. This

goes against the deterministic world assumption that is made in [21]. Such limitation occurs when the agent is moving so slowly that the new state differs very little from the previous one. Hence, the agent perceives both states as being the same. Currently, the agent only calculates the reward values when the level ends, so the problem is mitigated by only taking into account the most recent state of a series of consecutive equal states. However, by ignoring the other states, some important information can be lost. A way of avoiding this workaround, is to increase the number of features captured by the game-state. However, by doing so, several different states that are very similar are going to be generated. If there is a huge number of possible states, then the learning process will be much slower.

7.3.3 Finding known states

The usage of two dictionary-like structures to store the agent knowledge-base in memory can become a bottleneck when the number of states increase. Currently, the state identifier is a string describing the features that are captured so as to easy the debug task. However, the string comparison process is slow and can consume all the time the agent has to play. In the future more intelligent ways of storing the knowledge, such as creating a multilevel dictionary or loading smaller learning files only when the agent need them, must be used in order to speed-up the decision process.

Chapter 8

Conclusions and Future Work

8.1 Conclusion

This dissertation's goal was to develop a solution that tackled the over-specialization of the agents to the public levels of Geometry Friends.

The proposed approach is based on a Divide-and-Conquer strategy that splits the GF problem into three sub-problems. To solve those three problems algorithms of reinforcement learning and searching were used.

The tests made with the circle agent against the competitors of the 2014 edition's of GF show that the developed solution performed better than the others in the private levels of that competition. Despite the fact that the agent would get a final score much lower than CIBot's agent on the competition, the results ratify that the proposed solution successfully overcome the over-specialization problem reported on previous solutions for GF as it displayed a constant behaviour on both public and private level sets. Moreover, the agent won a greater number of levels than any other of the agents that were submitted to the competition. The strategy presented for the circle agent still has some problems that once fixed, can improve the score of the agent in the competition, therefore making it a respectable challenger for the first place on future editions of the competition.

This dissertation, together with the work of [32] makes it clear that Reinforcement Learning can be successfully used to solve the GF problem. This work leaves basis for future work on GF, both on the improvement if the quality of the circle agent or on the implementation of the same strategy on the rectangle agent.

8.2 Future Work

The next step of this work is to apply the same strategy to the rectangle agent. To implement the same method on that agent, some modifications have to be made to this solution. The division of the platforms has to take into account the specific movements of the character. Because the rectangle can change its shape, there are some platforms that are only accessible when the character has a specific shape.

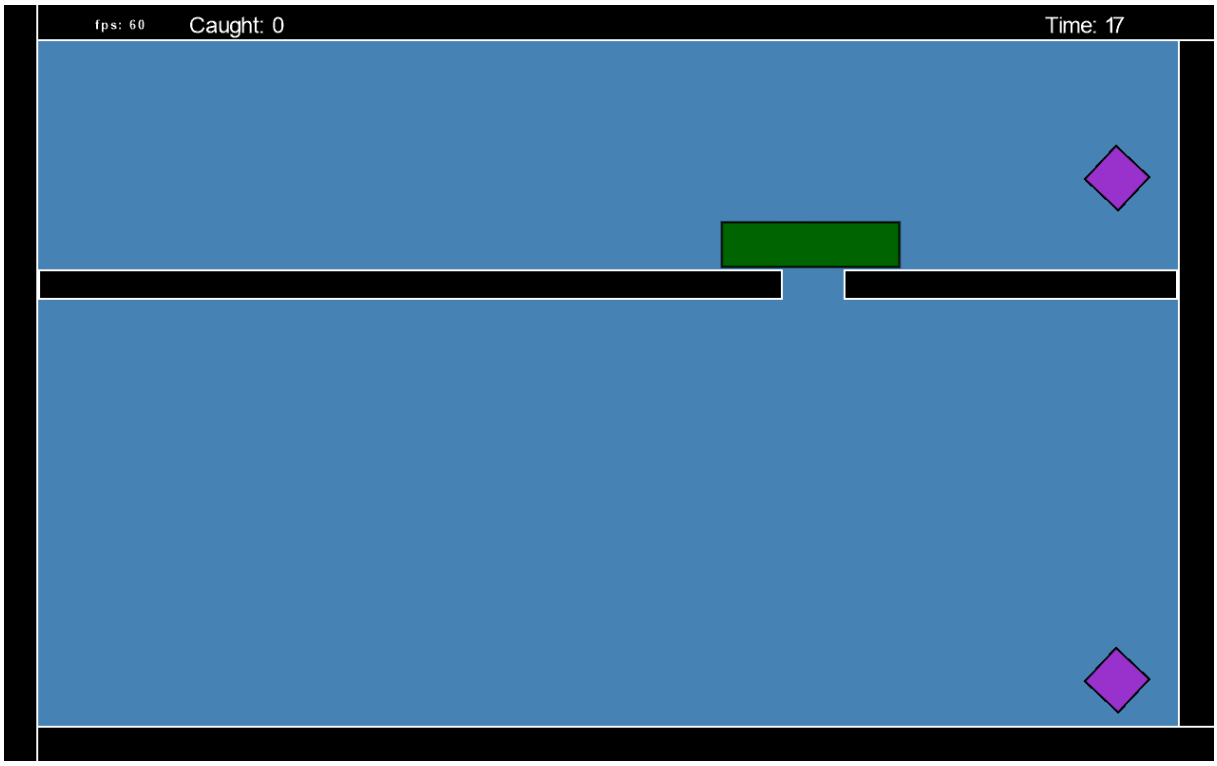


Figure 8.1: Problem when rectangle is in the middle of two platforms. The approach proposed in this work states that an agent is either on a platform or not on a platform at all. In this case, the rectangle is on two platforms. The rectangle agent solution must be aware of this challenge.

Moreover, when solving SP3, the agent has also to take into account the shape of the character, as the it is the critical feature to overcome some of the instances of this sub problem such as the one depicted on Figure 8.1.

After having both agents working with the approach proposed by this work, the problem of having the agents working on cooperative environments can be tackled. The evaluation of such agents can be made by comparing the results obtained with the results of the participants of the cooperative track of the competition. A first approach to the problem of cooperation can aim to create two agents that see the other one as another platform to where it can go to. This can impose several challenges due to the fact that this ”platform” is dynamic. This dynamism, in theory, can be very hard to capture by the Reinforcement Learning algorithms. As both agents are moving simultaneously, several outcomes will be registered for the same pair $< state, action >$. Nevertheless, the emergence of cooperation from such scenarios can be tested. To tackle the challenges imposed by the cooperative environment, it is important to infer what are the intentions of the other agent by observing its movements.

Even after the cooperation problem is solved, there is still development that can be done in GF to equip the agents with skills to play alongside humans. Although this situation can be seen as another example of cooperation, the truth is that there is a higher degree of uncertainty when playing with humans. The agent has to be able to adapt to the different strategies used by the humans as well as to be able to deal with different degrees of proficiency on GF that the human players can have.

Bibliography

[1] A. M. Turing, "Computing machinery and intelligence," *Mind*, pp. 433–460, 1950.

[2] F. Hsu, *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2002.

[3] J. Schaeffer, N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen, "Checkers is solved," *Science*, vol. 317, no. 5844, pp. 1518–1522, 2007.

[4] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. Murdock, E. Nyberg, J. Prager, N. Schlaefer, and C. Welty, "Building WATSON: An overview of the DeepQA project," *AI Magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[5] P. Sweetser and J. Wiles, "Current ai in games : a review," *Australian Journal of Intelligent Information Processing Systems*, vol. 8, no. 1, pp. 24–42, 2002. [Online]. Available: http://eprints.qut.edu.au/45741/

[6] S. Rabin, "Designing a general robust ai engine," *Game Programming Gems*, vol. 1, pp. 221–236, 2000.

[7] A. LaMothe, *Tricks of the Windows game programming gurus*. Sams, 1999.

[8] S. Karakovskiy and J. Togelius, "The mario ai benchmark and competitions," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 55–67, March 2012.

[9] S. M. Lucas, "Evolving a neural network location evaluator to play ms. pac-man." in *CIG*. Citeseer, 2005.

[10] P. Hingston, "A turing test for computer game bots," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 1, no. 3, pp. 169–186, 2009.

[11] ——, "A new design for a turing test for bots," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, Aug 2010, pp. 345–350.

[12] B. G. Weber, P. Mawhorter, M. Mateas, and A. Jhala, "Reactive planning idioms for multi-scale game ai," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 115–122.

[13] J. Togelius, S. Karakovskiy, and R. Baumgarten, “The 2009 mario ai competition,” in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, July 2010, pp. 1–8.

[14] S. Bojarski and C. Congdon, “Realm: A rule-based evolutionary computation agent that learns to play mario,” in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, Aug 2010, pp. 83–90.

[15] N. C. Hou, N. S. Hong, C. K. On, and J. Teo, “Infinite mario bross ai using genetic algorithm,” in *Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2011 IEEE Conference on*. IEEE, 2011, pp. 85–89.

[16] S. Ross and J. Bagnell., “Efficient reductions for imitation learning,” in *Proc. 13th Int. Conf. Artificial Intelligence and Statistics*, 2010.

[17] J.-J. Tsay, C.-C. Chen, and J.-J. Hsu, “Evolving intelligent mario controller by reinforcement learning,” in *Technologies and Applications of Artificial Intelligence (TAAI), 2011 International Conference on*. IEEE, 2011, pp. 266–272.

[18] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, “Heads-up limit hold’em poker is solved,” *Science*, vol. 347, no. 6218, pp. 145–149, 2015.

[19] S. Gelly and D. Silver, “Achieving master level play in 9×9 computer Go,” in *Proc. 23rd AAAI Conf. Artificial Intelligence*, 2008, pp. 1537–1540.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” arXiv:1312.5602, 2013.

[21] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of artificial intelligence research*, pp. 237–285, 1996.

[22] L. Galway, D. Charles, and M. Black, “Improving temporal difference game agent control using a dynamic exploration during control learning,” in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009, pp. 38–45.

[23] S. Sen, M. Sekaran, and J. Hale, “Learning to coordinate without sharing information,” *Proceedings of the National Conference on Artificial Intelligence*, pp. 426–431, 1994. [Online]. Available: http://www.aaai.org/Papers/AAAI/1994/AAAI94-065.pdf

[24] C. Boutilier, “Planning, learning and coordination in multiagent decision processes,” in *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*. Morgan Kaufmann Publishers Inc., 1996, pp. 195–210.

[25] S. Russell, P. Norvig, and A. Intelligence, “A modern approach,” *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, 1995.

[26] J. B. Rocha, S. Mascarenhas, and R. Prada, “Game mechanics for cooperative games,” in *Zon Digital Games 2008*. Porto, Portugal: Centro de Estudos de Comunicação e Sociedade, Universidade do Minho, November 2008, pp. 72–80.

[27] J. B. G. Rocha, “Geometry friends,” Ph.D. dissertation, Master’s thesis, IST, University of Lisbon, 2009.

[28] C. Fraga, “Motion control for an artificial character teaming with a human player in a casual game,” Master’s thesis, Instituto Superior Tecnico, 2011.

[29] D.-M. Yoon and K.-J. Kim, “Cibot circle agent technical report,” http://gaips.inesc-id.pt:8081/geometryfriends/?page_id=476, 2014.

[30] D. A. Vallade Benoît and T. Nakashima, “Opu-scom technical report,” http://gaips.inesc-id.pt:8081/geometryfriends/?page_id=476, 2014.

[31] H.-T. Kim and K.-J. Kim, “Cibot square agent technical report,” http://gaips.inesc-id.pt:8081/geometryfriends/?page_id=476, 2014.

[32] L.-J. W. Yen-Wen Lin and T.-H. Chang, “Kuas-is lab technical report,” http://gaips.inesc-id.pt:8081/geometryfriends/?page_id=476, 2014.

[33] E. Charniak and R. P. Goldman, “A bayesian model of plan recognition,” *Artificial Intelligence*, vol. 64, no. 1, pp. 53–79, 1993.

[34] H. Bui, “A general model for online probabilistic plan recognition,” in *In Proc. of the International Joint Conference on Artificial Intelligence (IJCAI*, 2003, pp. 1309–1315.

[35] N. Lesh, C. Rich, and C. L. Sidner, “Using plan recognition in human-computer collaboration,” *COURSES AND LECTURES-INTERNATIONAL CENTRE FOR MECHANICAL SCIENCES*, pp. 23–32, 1999.

Appendix A

Knowledge file

A excerpt of the knowledge file. To improve readability, the Excel file was exported as XML. For each row, the cells represent the following: 1st cell: State Id, 2nd cell: Roll Left Reward, 3rd cell: Roll Right Reward, 4th cell: Jump Reward.


```
<Row>
<Cell><Data ss:Type="String">/-Pos:7/+Pos:22/Speed:0/Colls:3/ColVector:[2/0]</Data></Cell>
<Cell><Data ss:Type="Number">531.89434068400499</Data></Cell>
<Cell><Data ss:Type="Number">465.57288517945898</Data></Cell>
<Cell><Data ss:Type="Number">390.09648679628799</Data></Cell>
</Row>
<Row>
<Cell><Data ss:Type="String">/-Pos:7/+Pos:22/Speed:1/Colls:3/ColVector:[2/0]</Data></Cell>
<Cell><Data ss:Type="Number">550.718793212018</Data></Cell>
<Cell><Data ss:Type="Number">523.85389118529304</Data></Cell>
<Cell><Data ss:Type="Number">91.816027570981404</Data></Cell>
</Row>
<Row>
<Cell><Data ss:Type="String">/-Pos:7/+Pos:22/Speed:2/Colls:3/ColVector:[2/0]</Data></Cell>
<Cell><Data ss:Type="Number">567.31698605470501</Data></Cell>
<Cell><Data ss:Type="Number">508.96146567070298</Data></Cell>
<Cell><Data ss:Type="Number">0</Data></Cell>
</Row>
<Row>
<Cell><Data ss:Type="String">/-Pos:8/+Pos:21/Speed:5/Colls:2/ColVector:[-7/0]/+Wall</Data></Cell>
<Cell><Data ss:Type="Number">1031.19874355556</Data></Cell>
<Cell><Data ss:Type="Number">1452.5260755675799</Data></Cell>
<Cell><Data ss:Type="Number">2349.31973080441</Data></Cell>
</Row>
```

Appendix B

Training levels for SP1

This appendix shows the layouts of the levels used on the first phase of the training process.

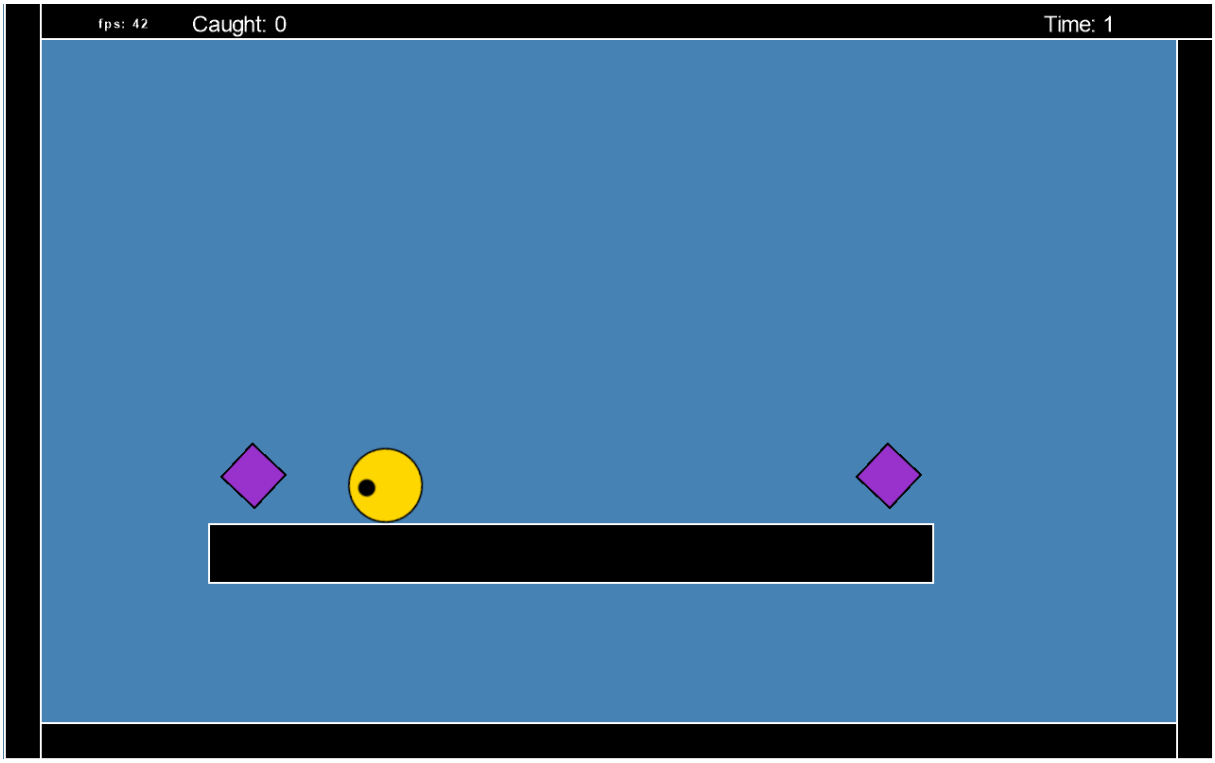


Figure B.1: Training Levels for SP1 – Level 1

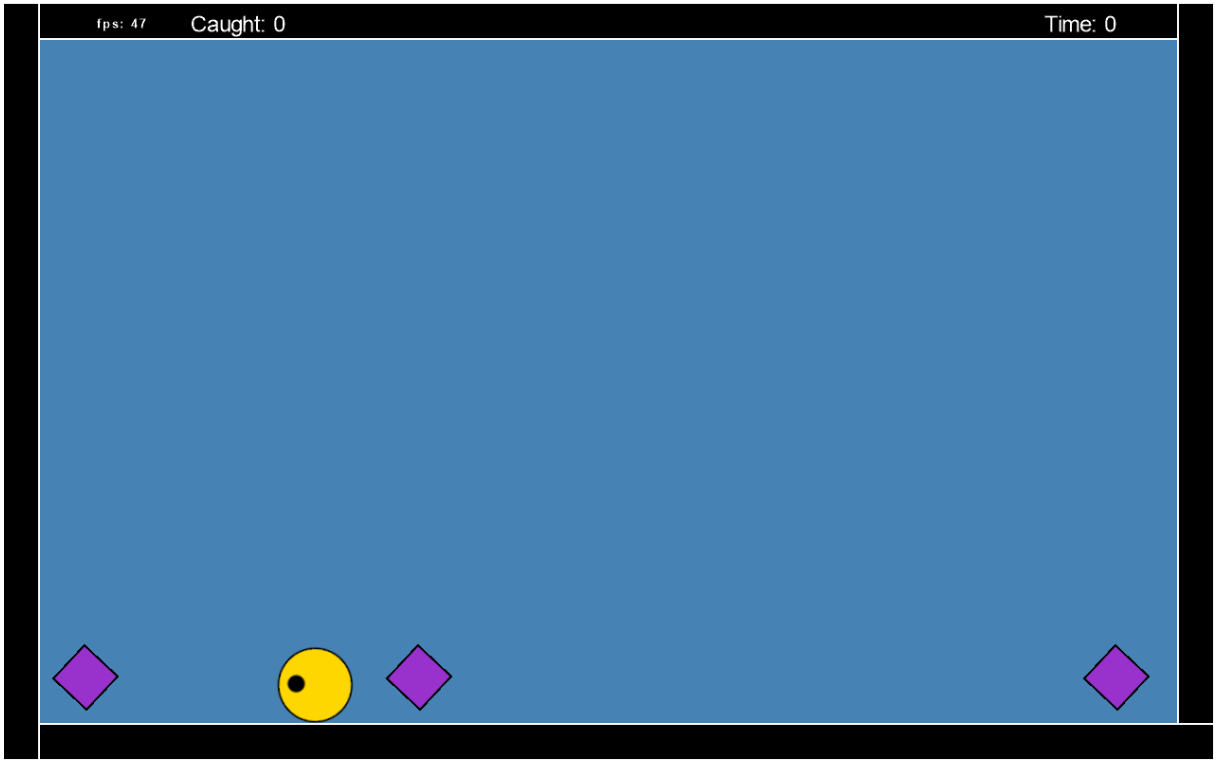


Figure B.2: Training Levels for SP1 – Level 2

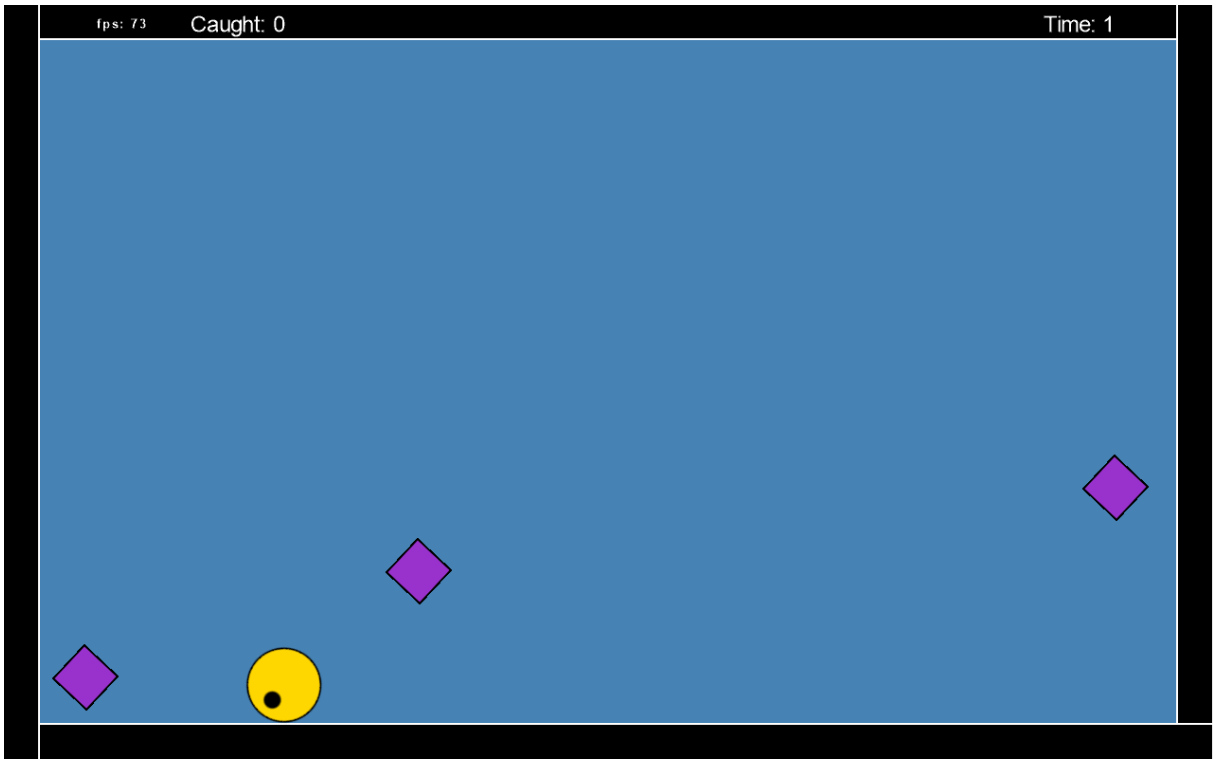


Figure B.3: Training Levels for SP1 – Level 3

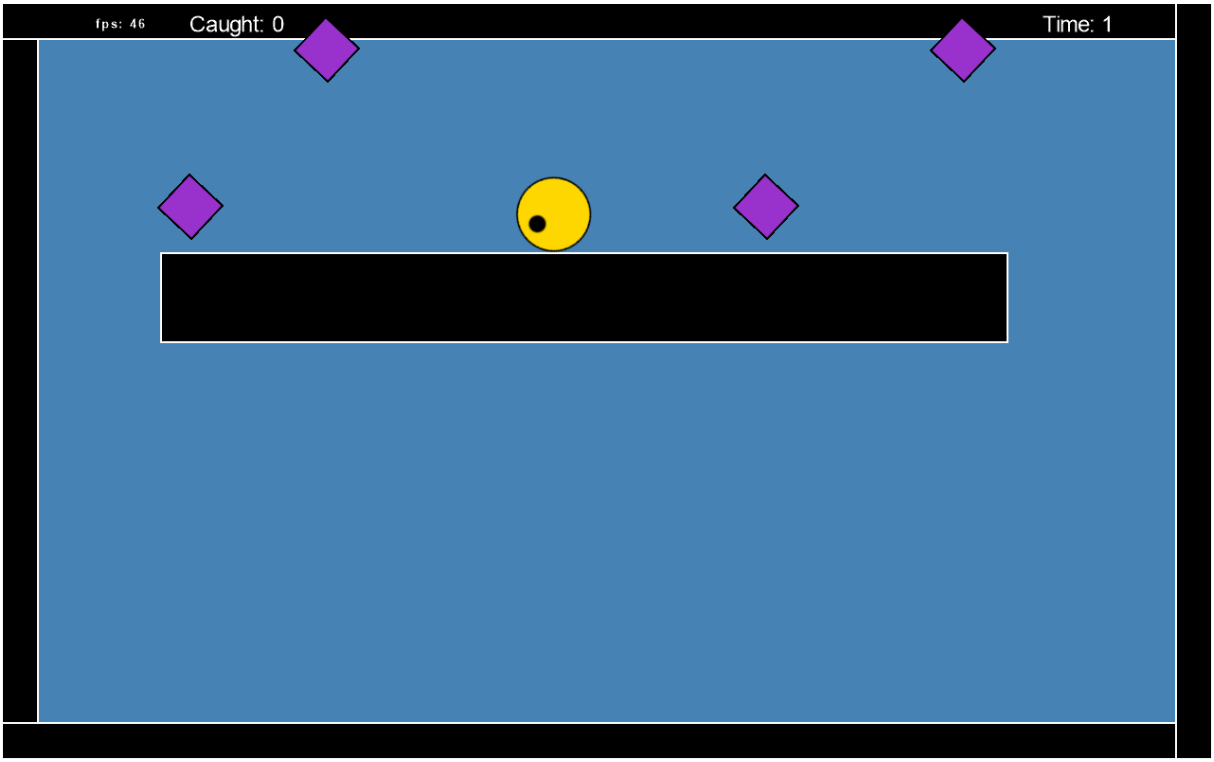


Figure B.4: Training Levels for SP1 – Level 4

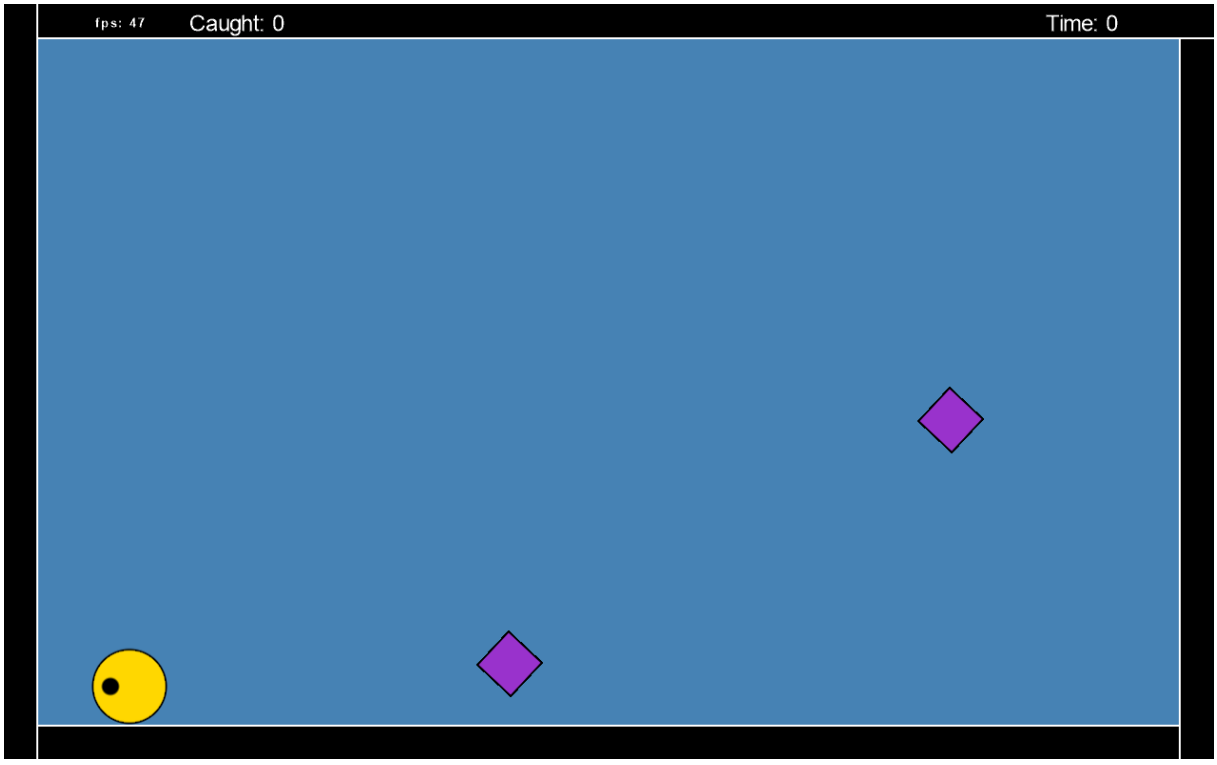


Figure B.5: Training Levels for SP1 – Level 5

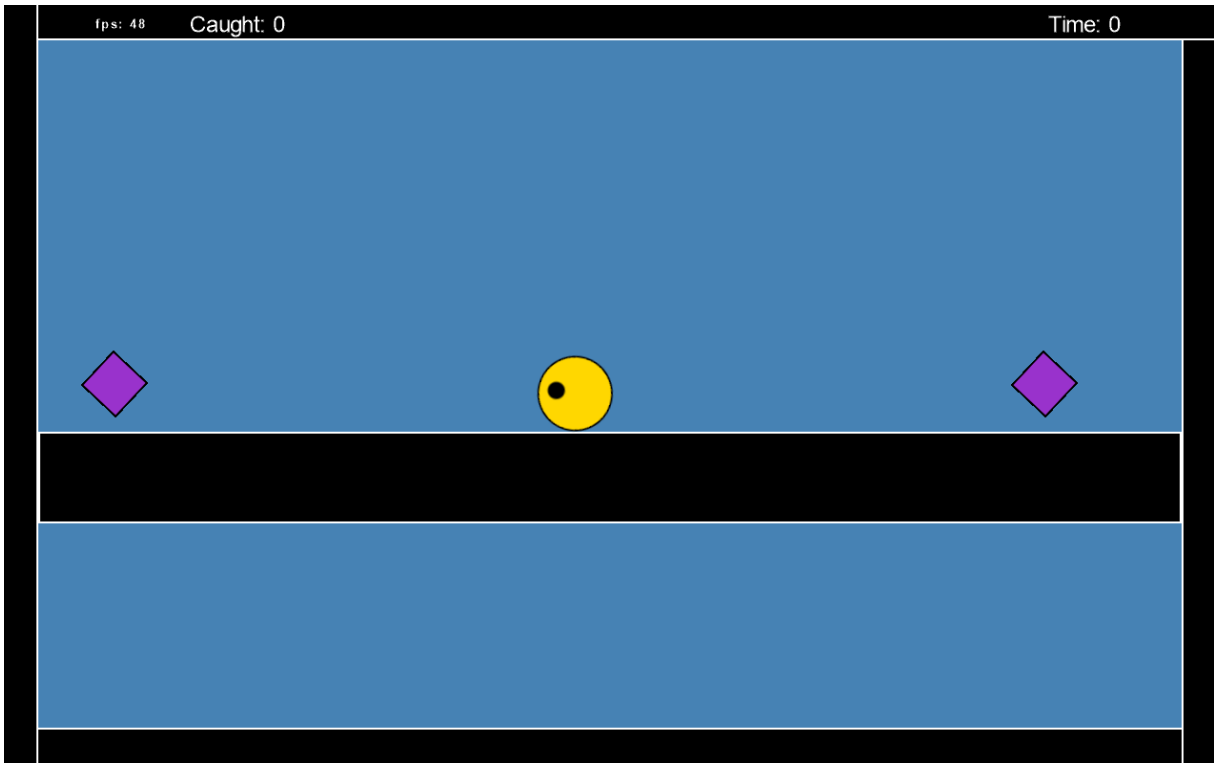


Figure B.6: Training Levels for SP1 – Level 6

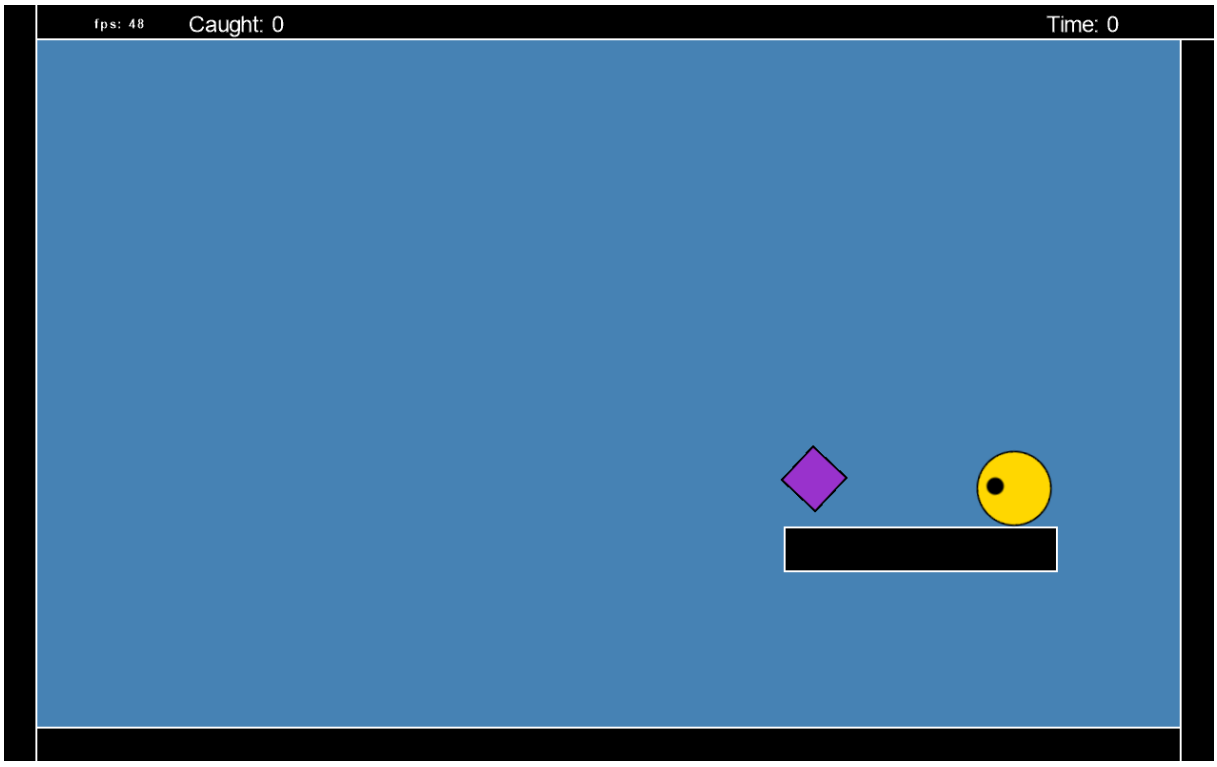


Figure B.7: Training Levels for SP1 – Level 7

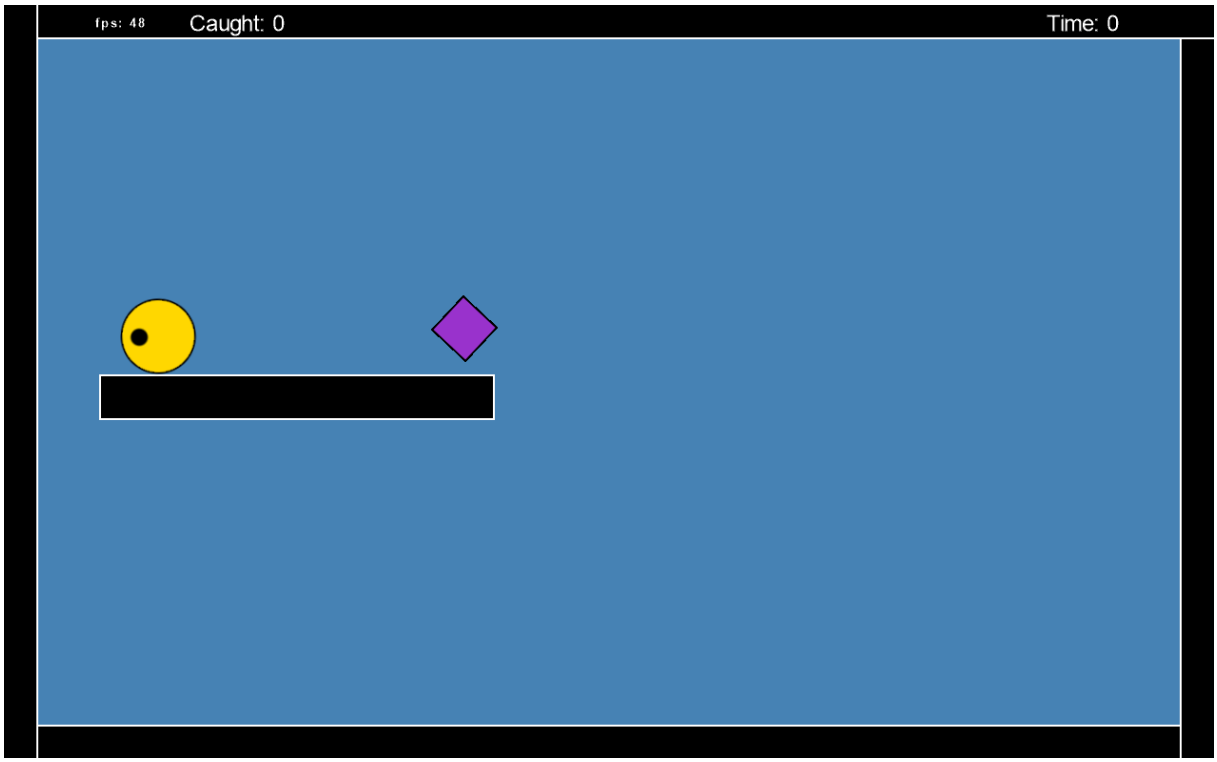


Figure B.8: Training Levels for SP1 – Level 8

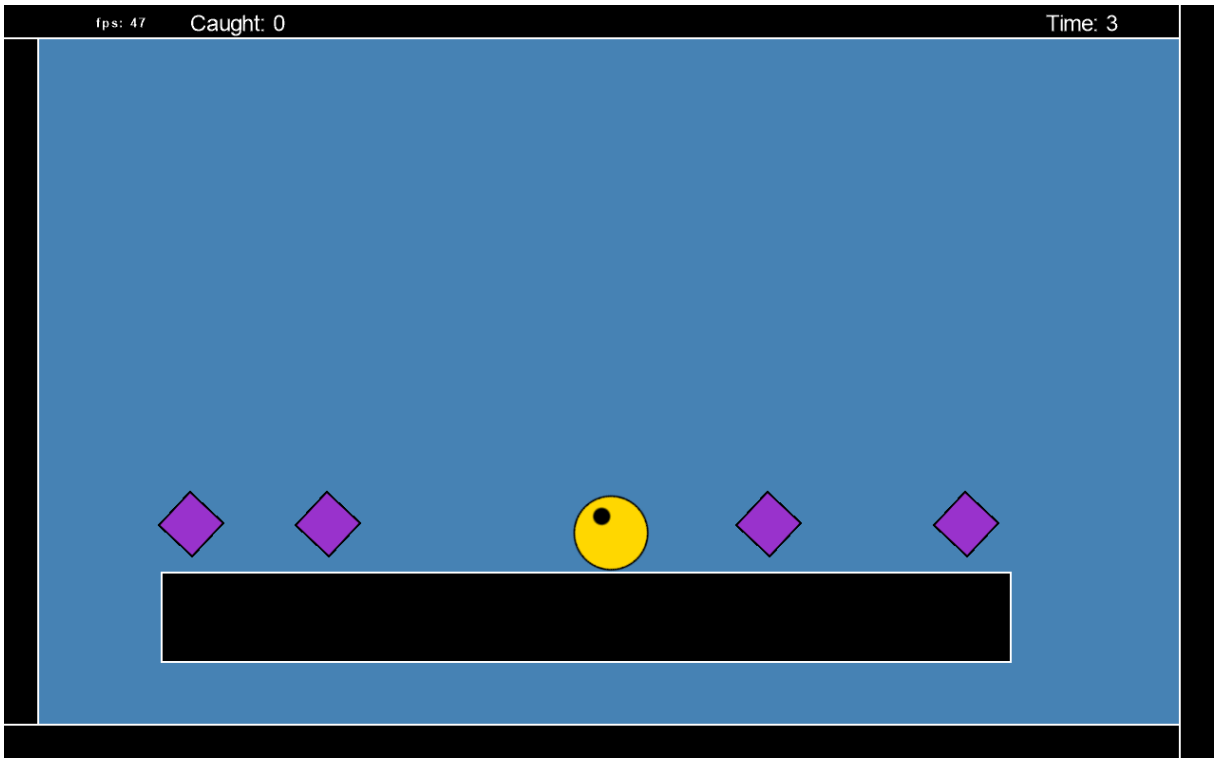


Figure B.9: Training Levels for SP1 – Level 9

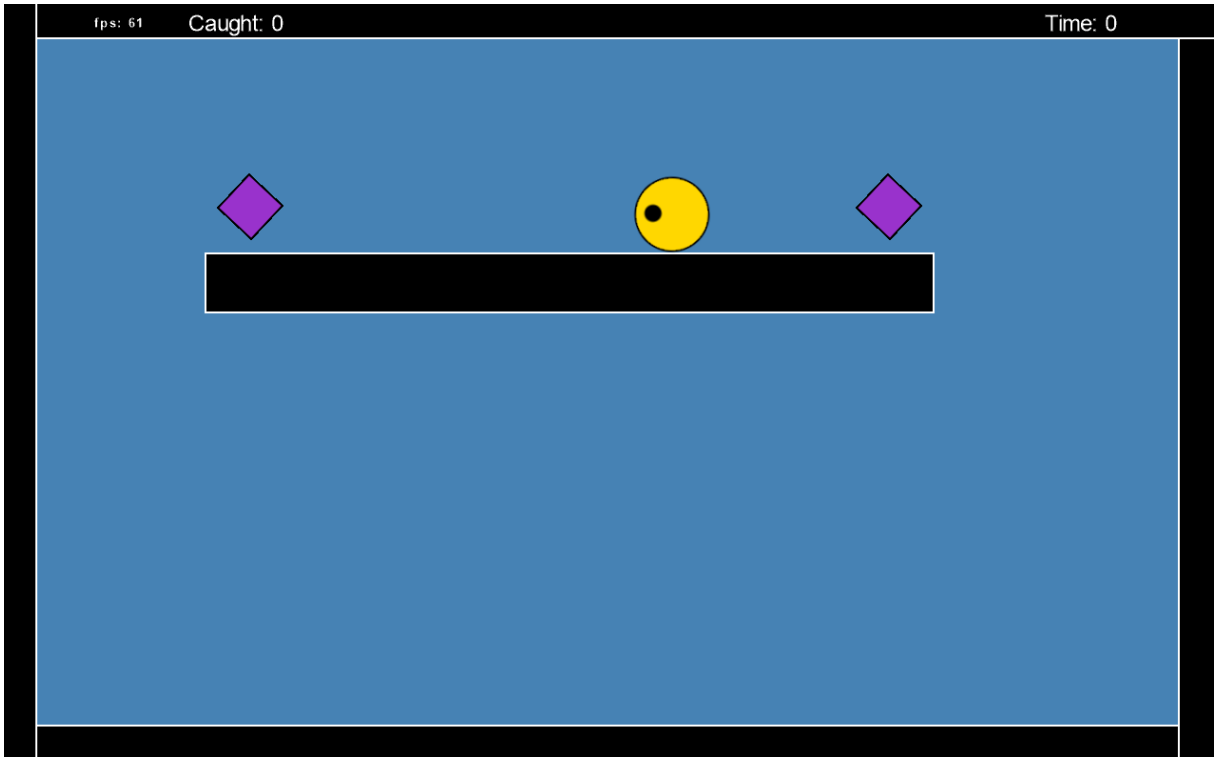


Figure B.10: Training Levels for SP1 – Level 10

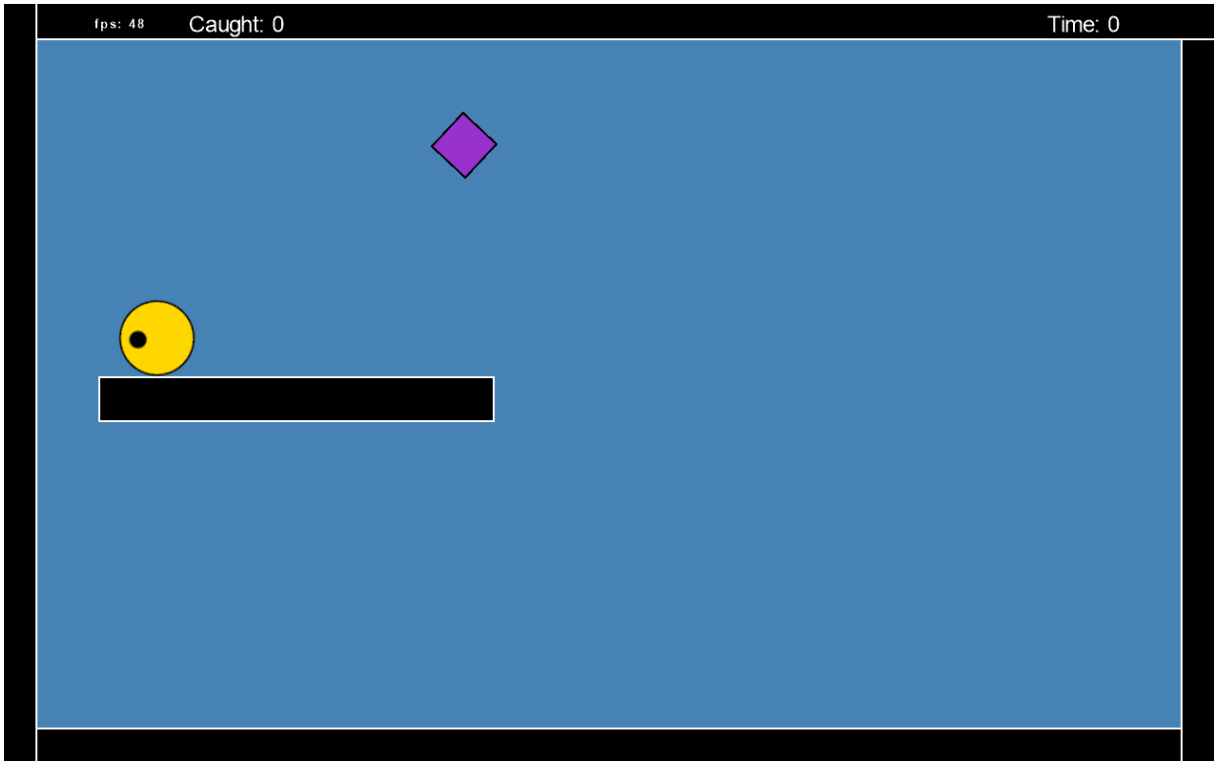


Figure B.11: Training Levels for SP1 – Level 11

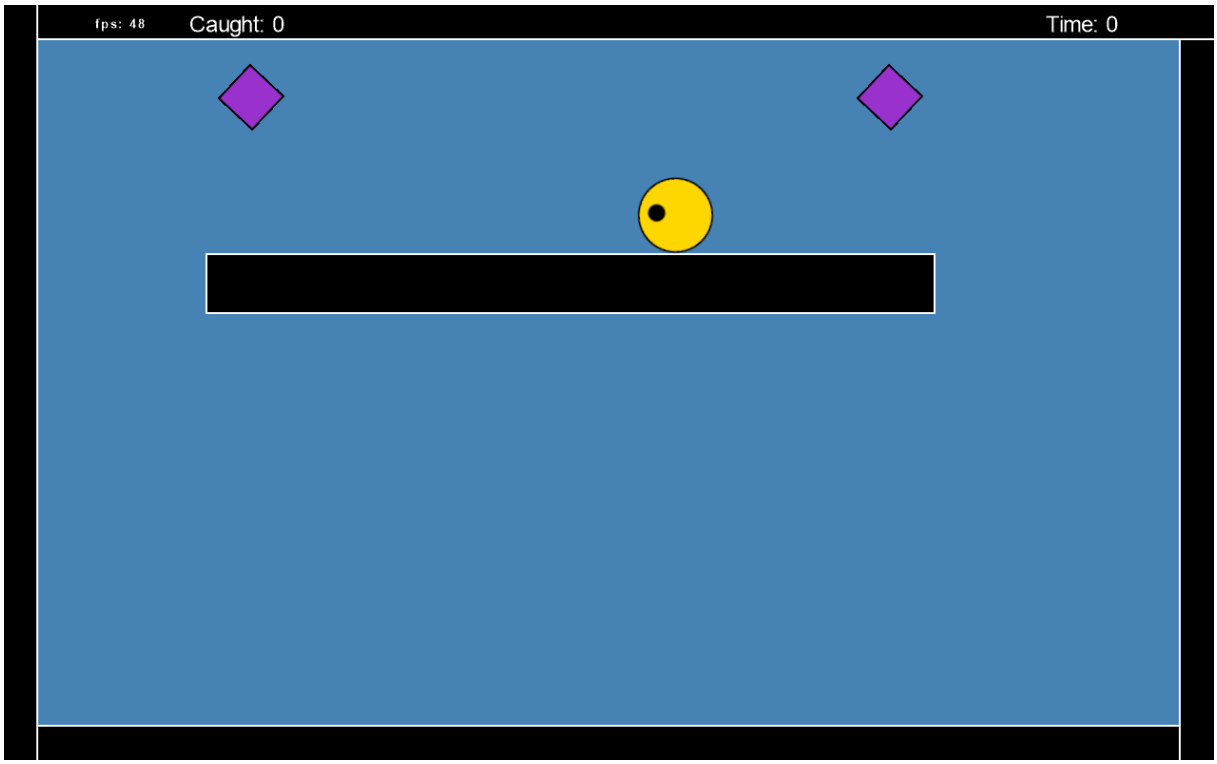


Figure B.12: Training Levels for SP1 – Level 12

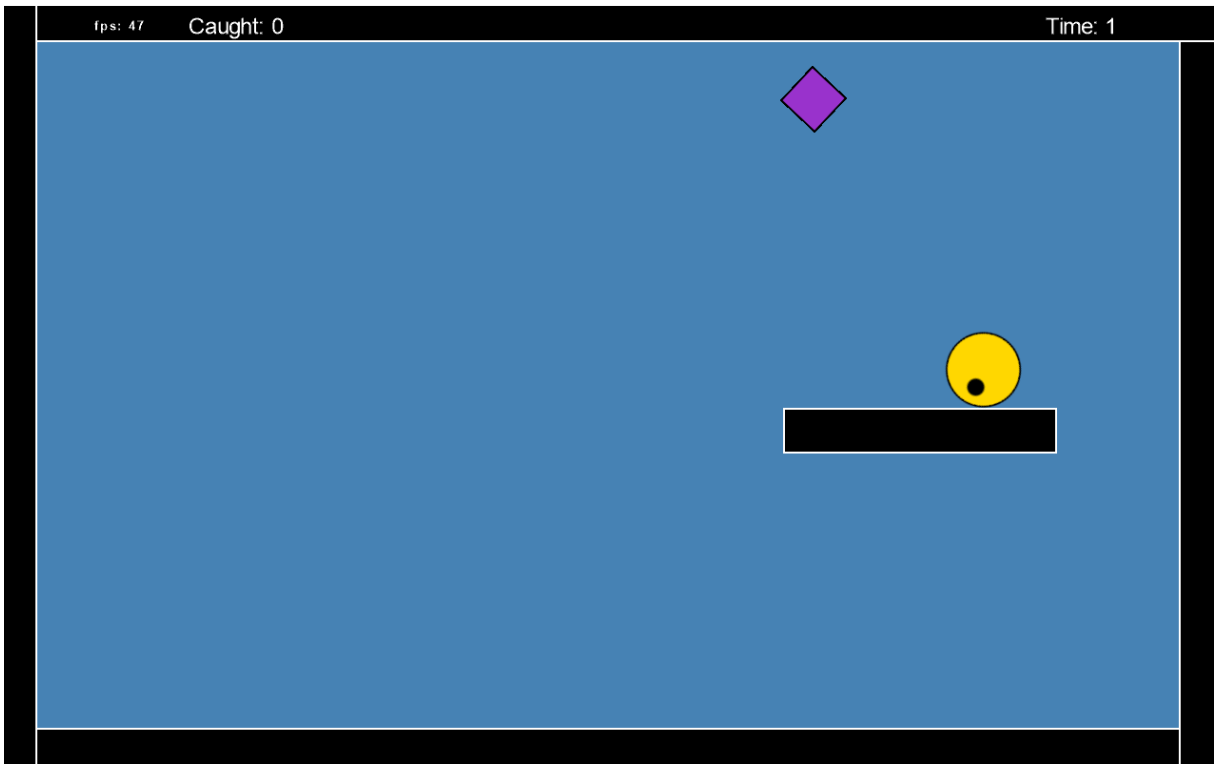


Figure B.13: Training Levels for SP1 – Level 13

Appendix C

Training levels for SP3

This appendix shows the layouts of the levels used on the second phase of the training process.

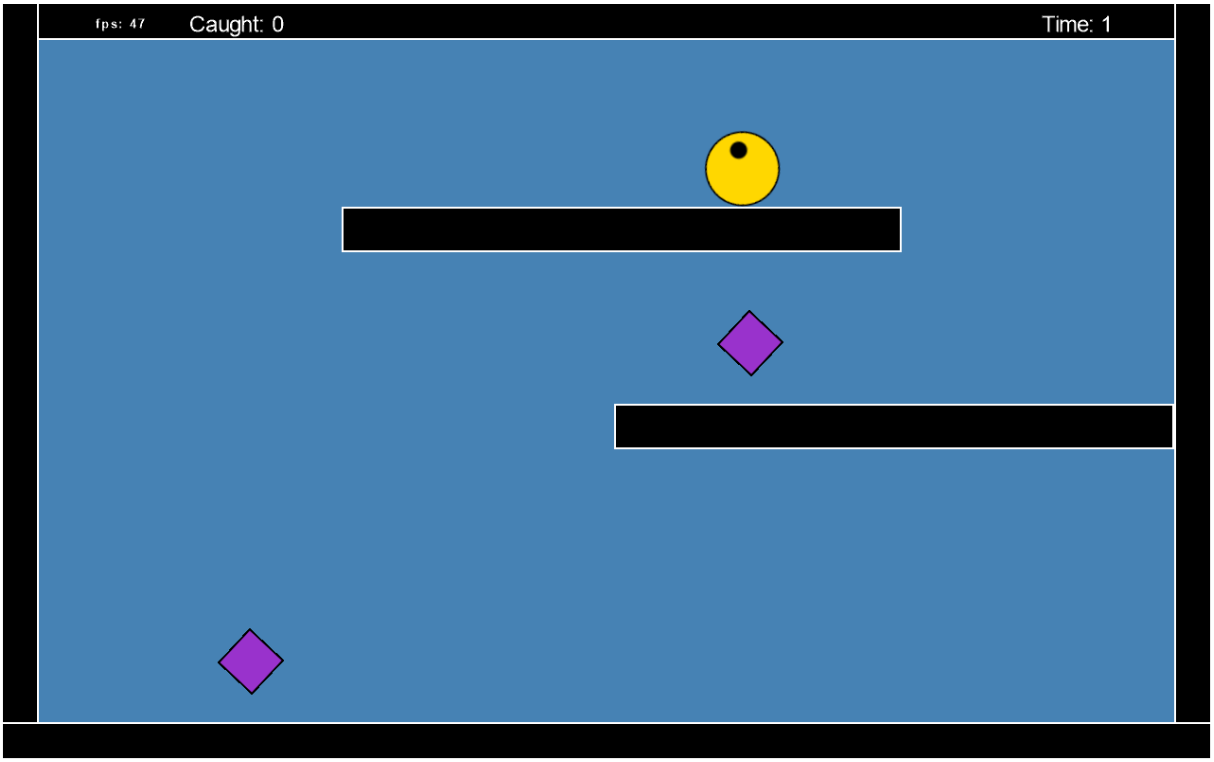


Figure C.1: Training Levels for SP3 – Level 1

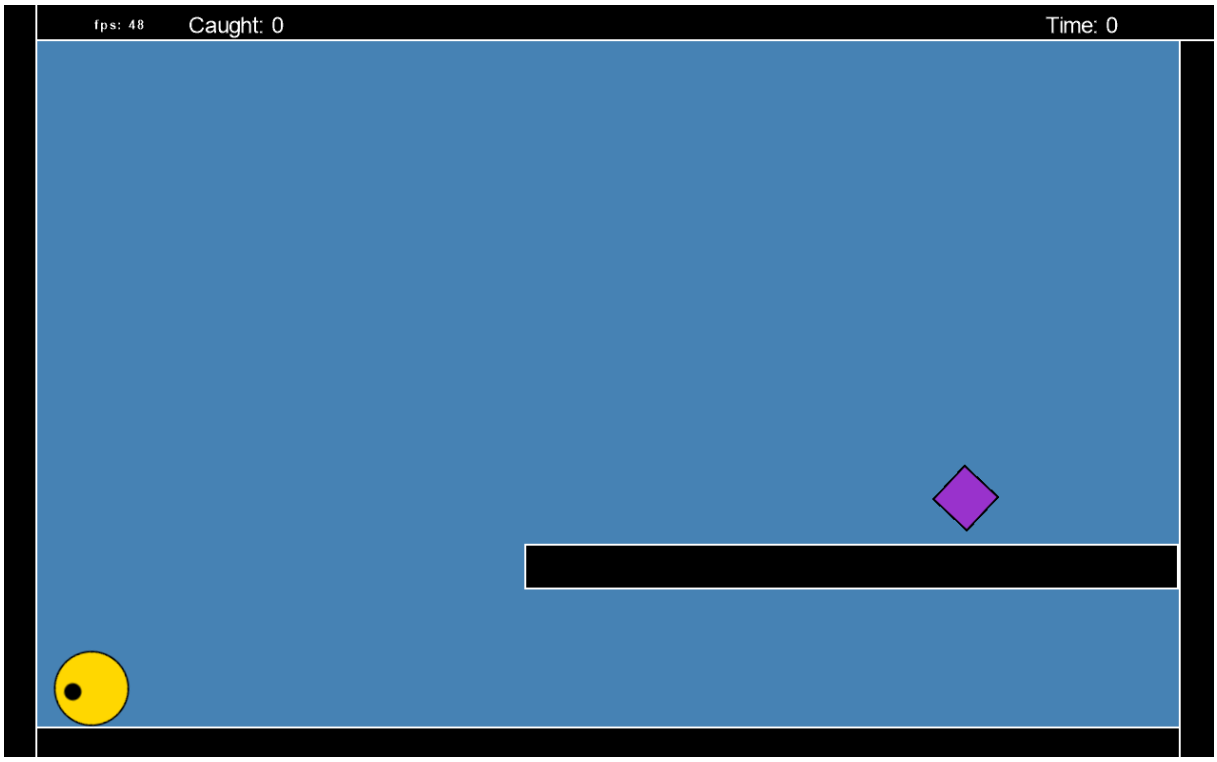


Figure C.2: Training Levels for SP3 – Level 2

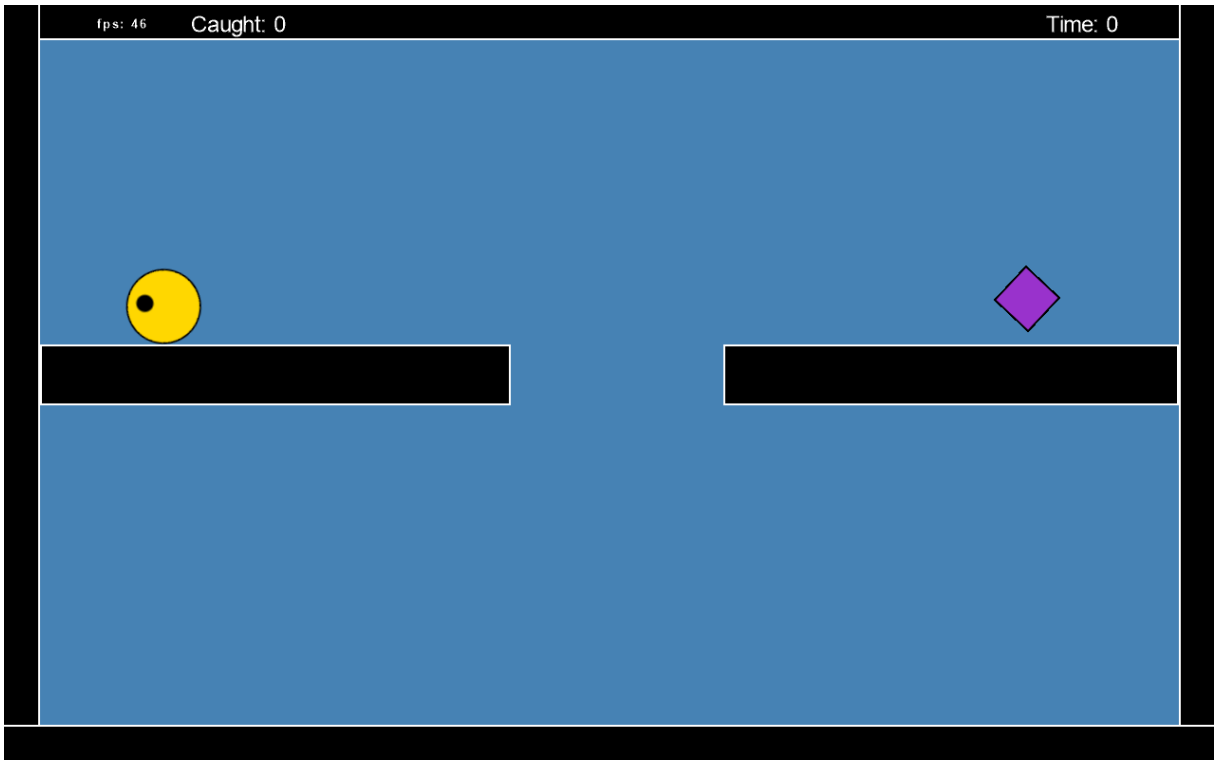


Figure C.3: Training Levels for SP3 – Level 3

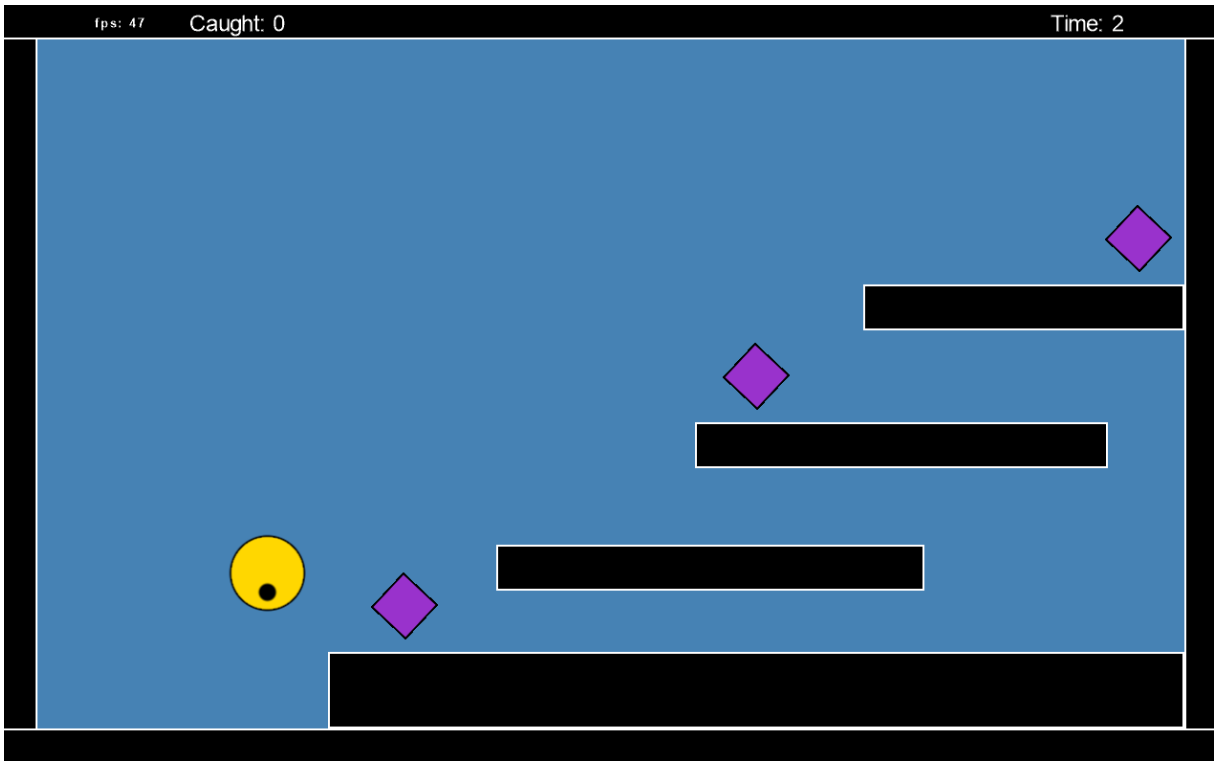


Figure C.4: Training Levels for SP3 – Level 4

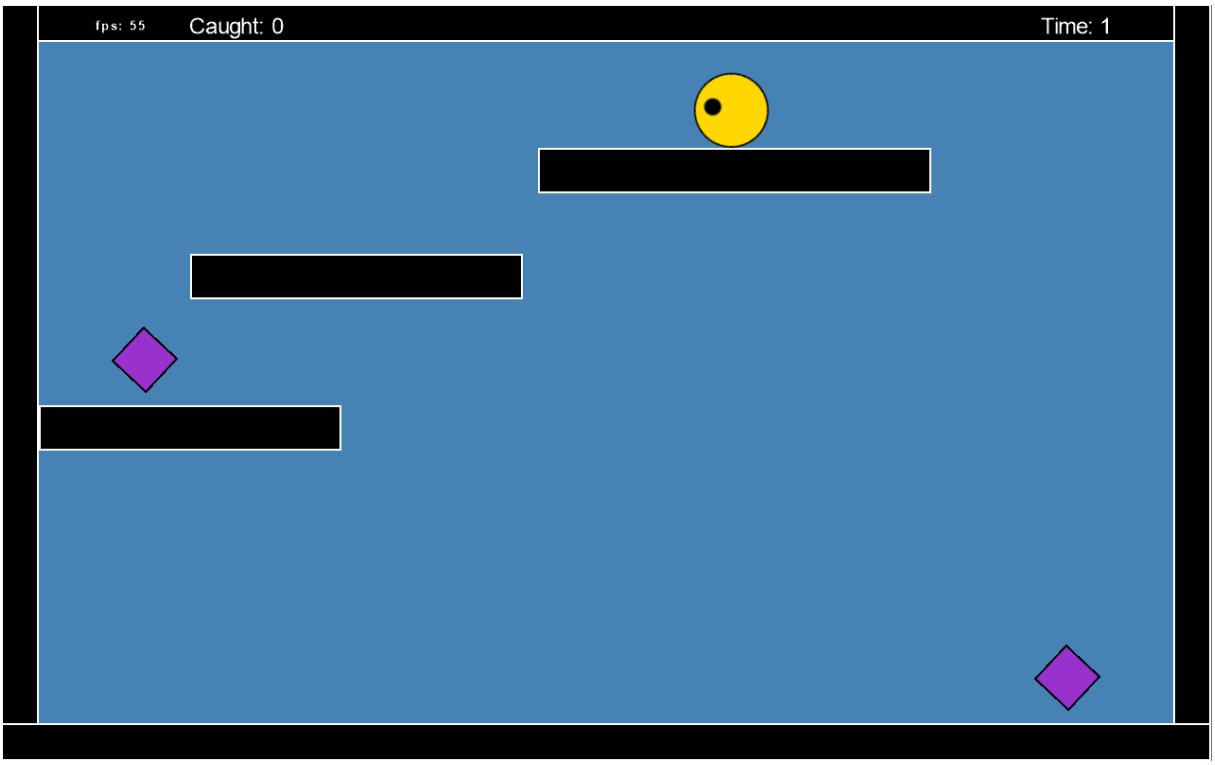


Figure C.5: Training Levels for SP3 – Level 5

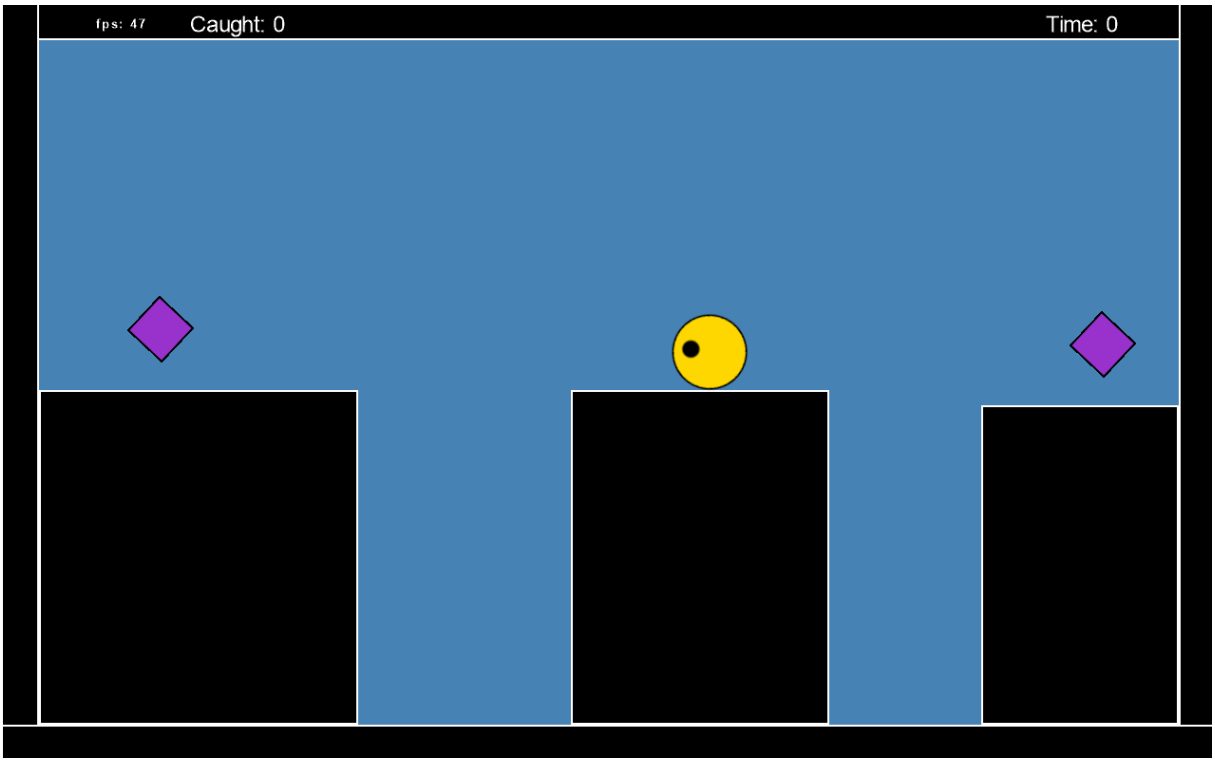


Figure C.6: Training Levels for SP3 – Level 6

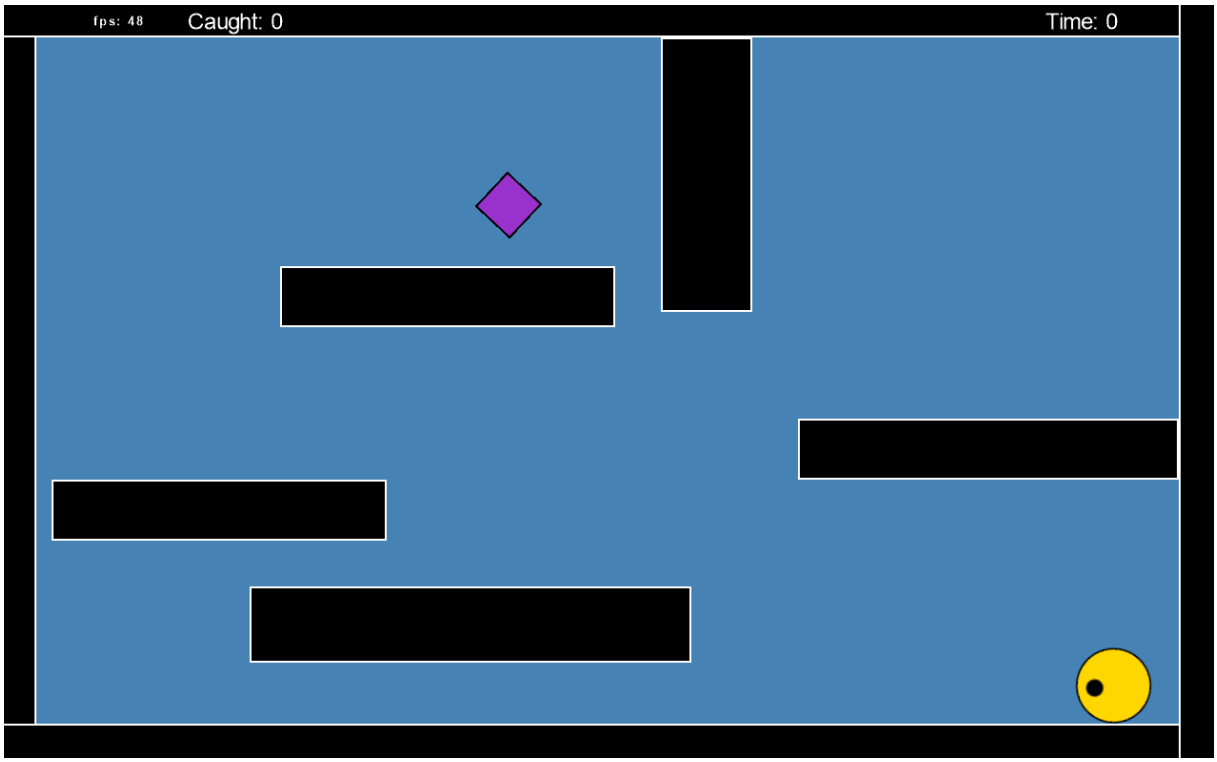


Figure C.7: Training Levels for SP3 – Level 7

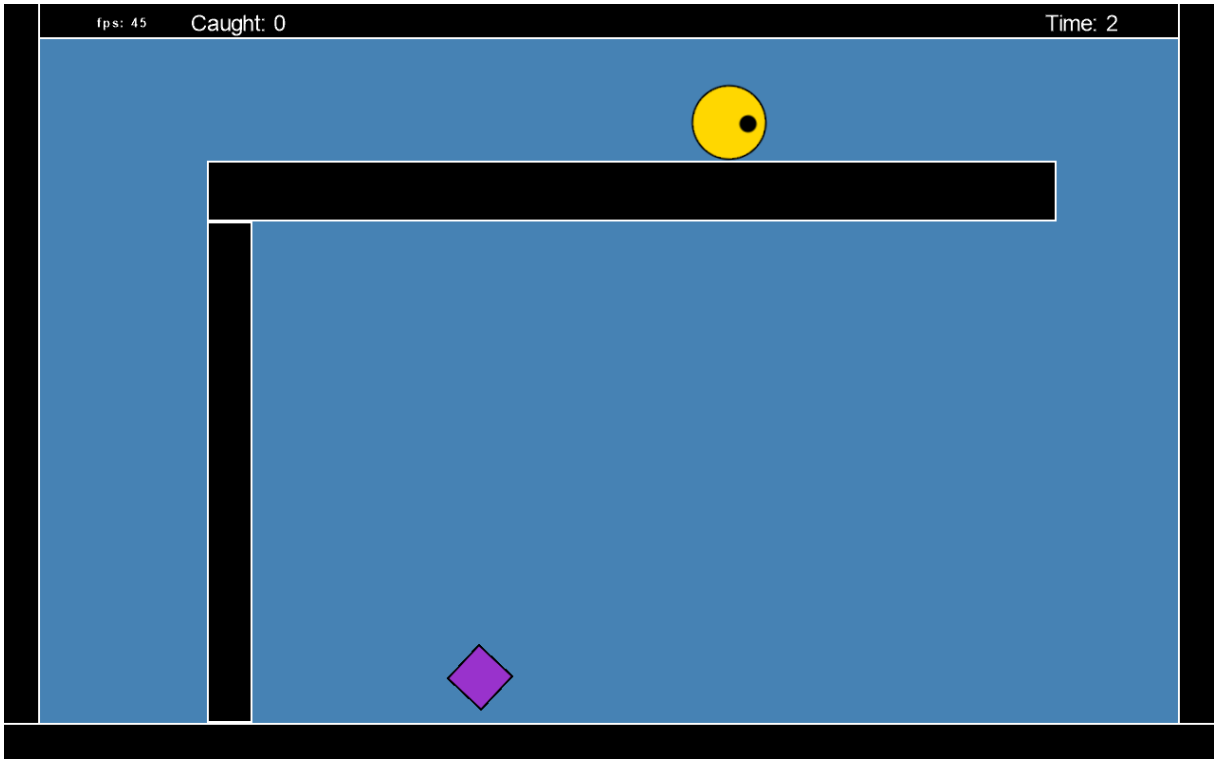


Figure C.8: Training Levels for SP3 – Level 8

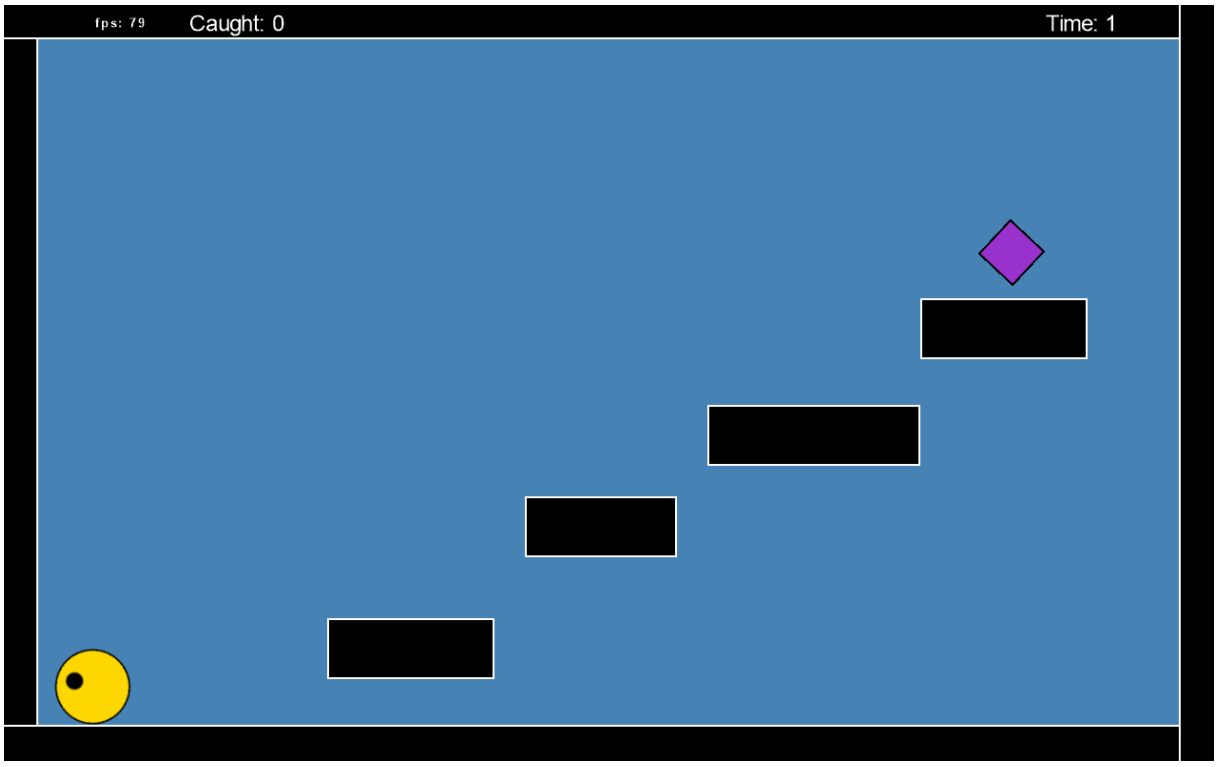


Figure C.9: Training Levels for SP3 – Level 9

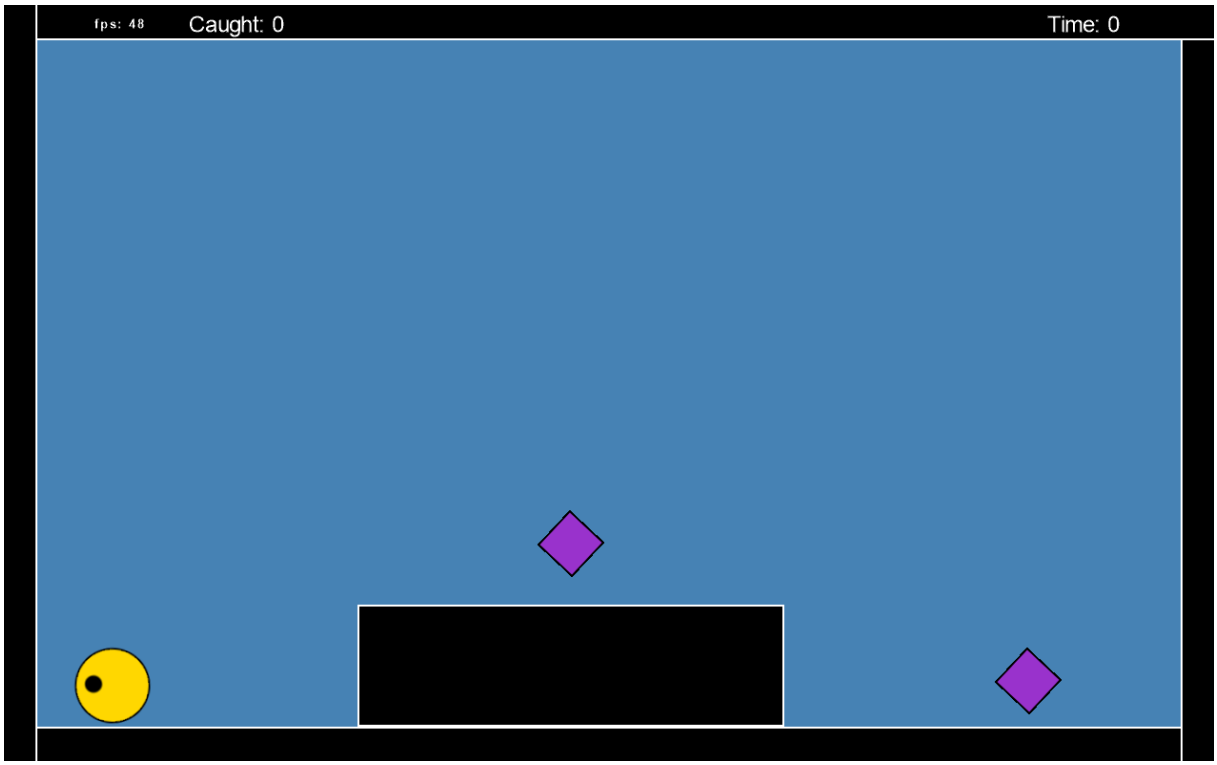


Figure C.10: Training Levels for SP3 – Level 10

Appendix D

GF 2014 Circle AI Competition Levels

This appendix shows the layouts of the ten levels used on the 2014 Geometry Friends AI Competition. This level set is used to evaluate the agent implemented in this work. Together with the levels presented on appendix B and on appendix C, they form the full training set that was used on the third phase of the training process.

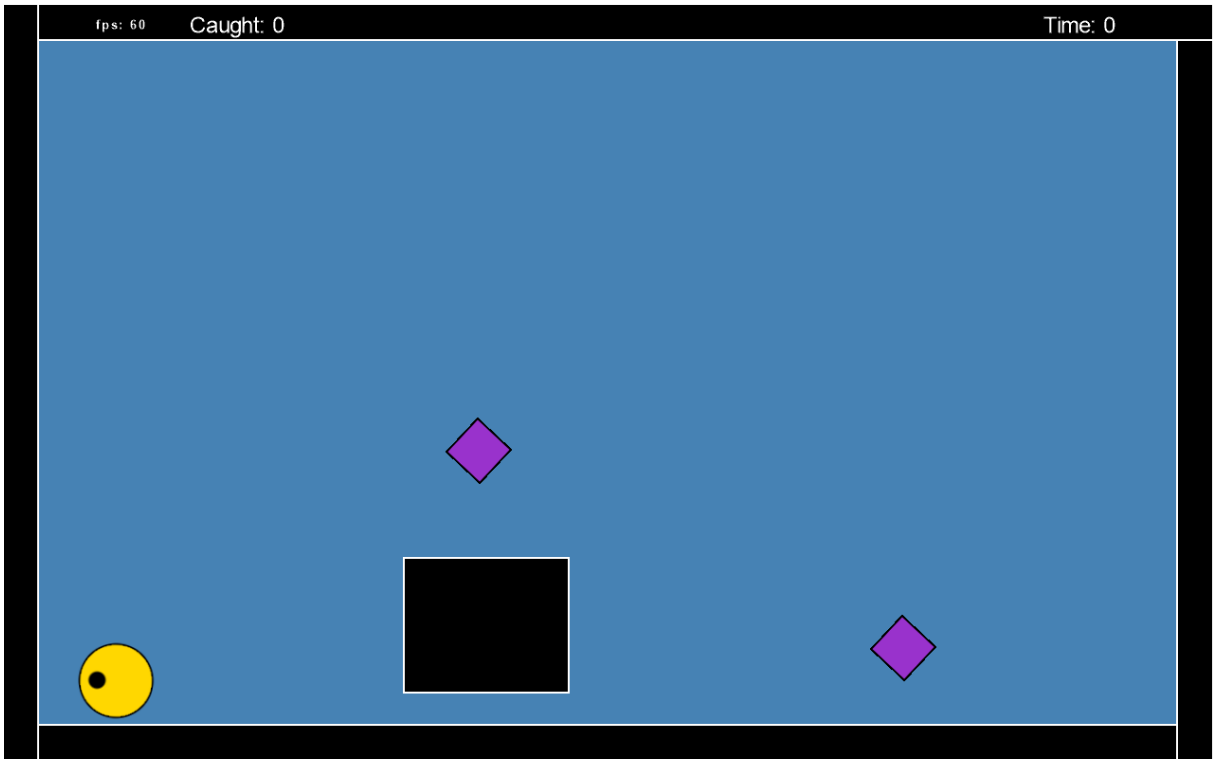


Figure D.1: Geometry Friends 2014 Circle AI Competition – Level 1

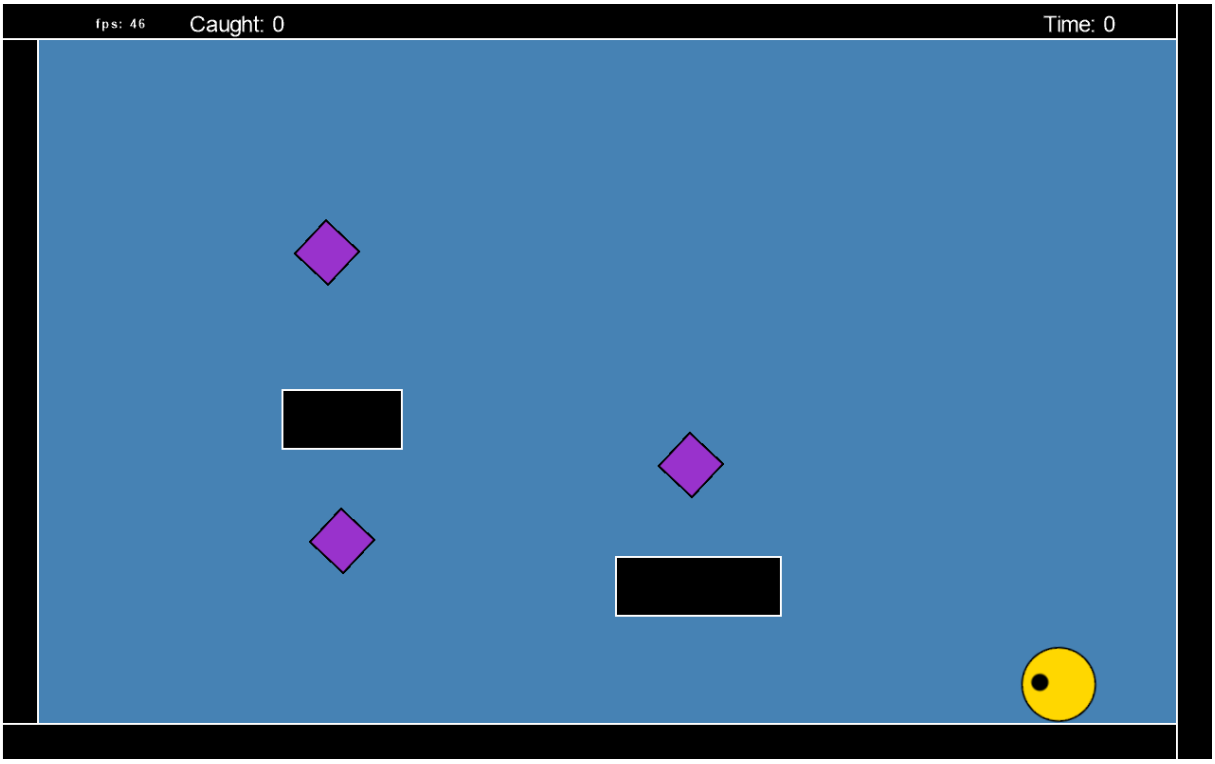


Figure D.2: Geometry Friends 2014 Circle AI Competition – Level 2

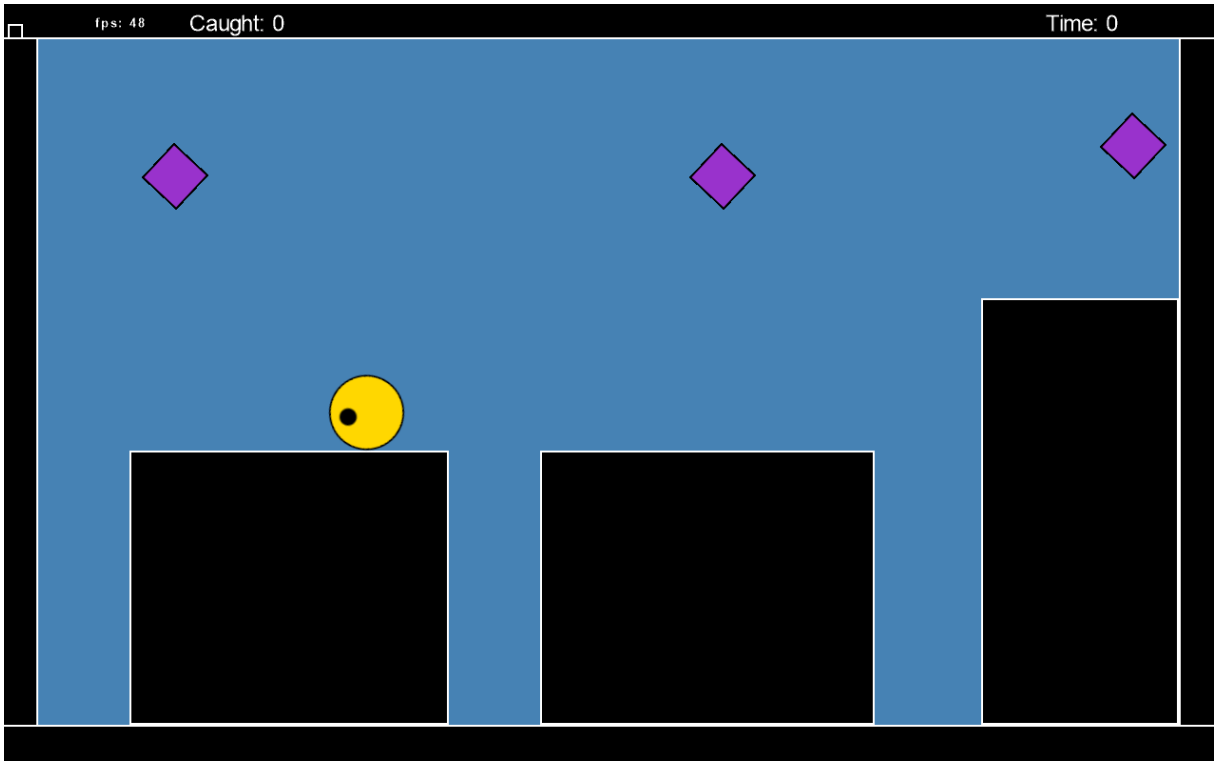


Figure D.3: Geometry Friends 2014 Circle AI Competition – Level 3

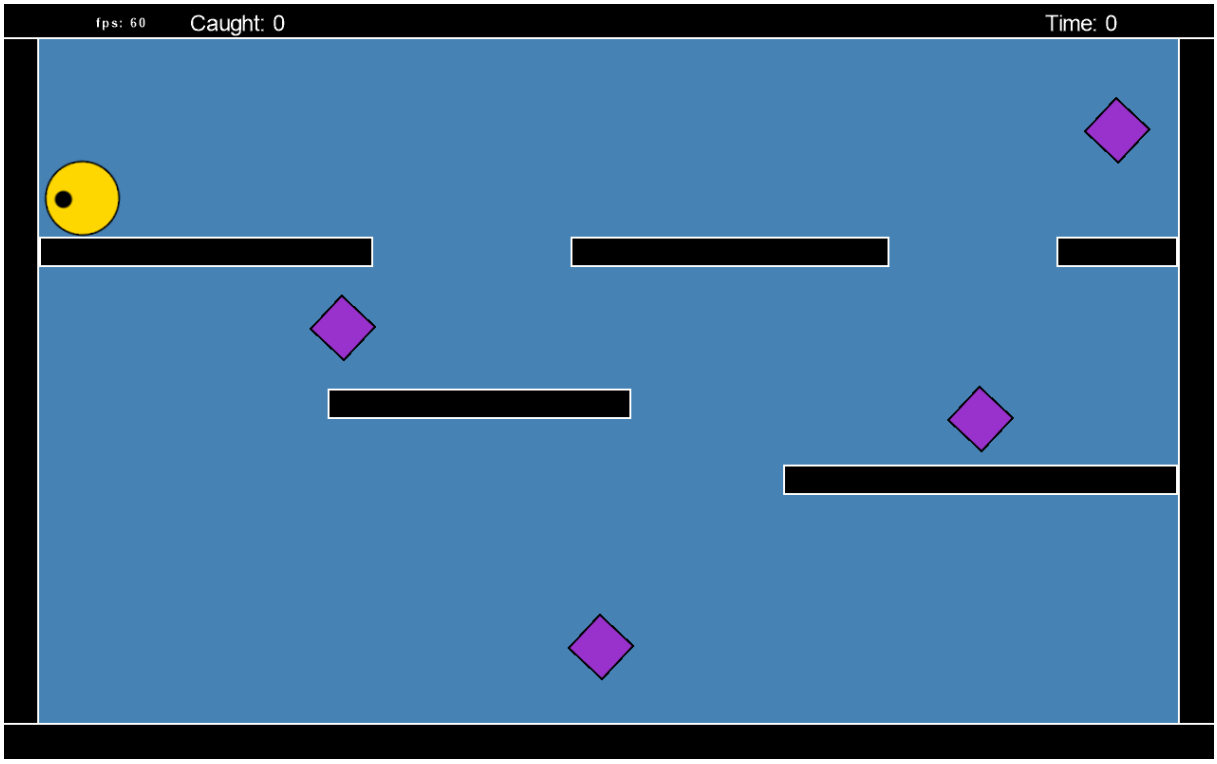


Figure D.4: Geometry Friends 2014 Circle AI Competition – Level 4

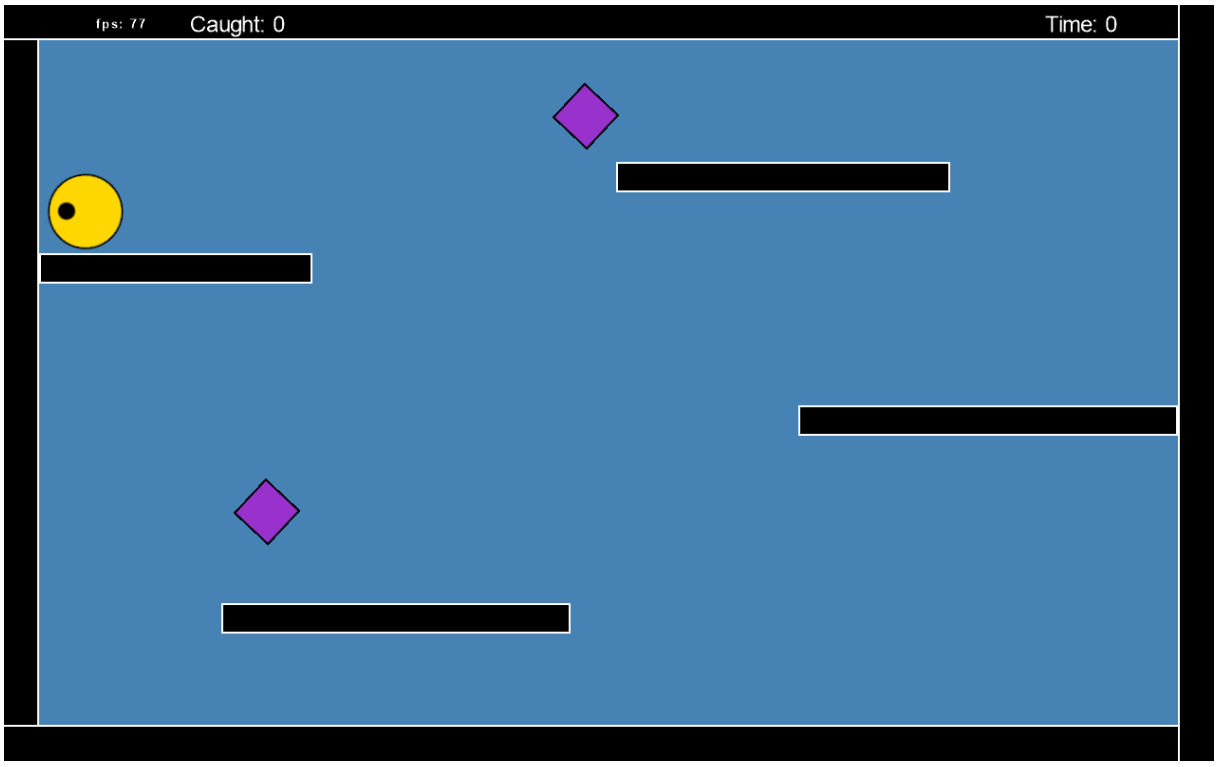


Figure D.5: Geometry Friends 2014 Circle AI Competition – Level 5

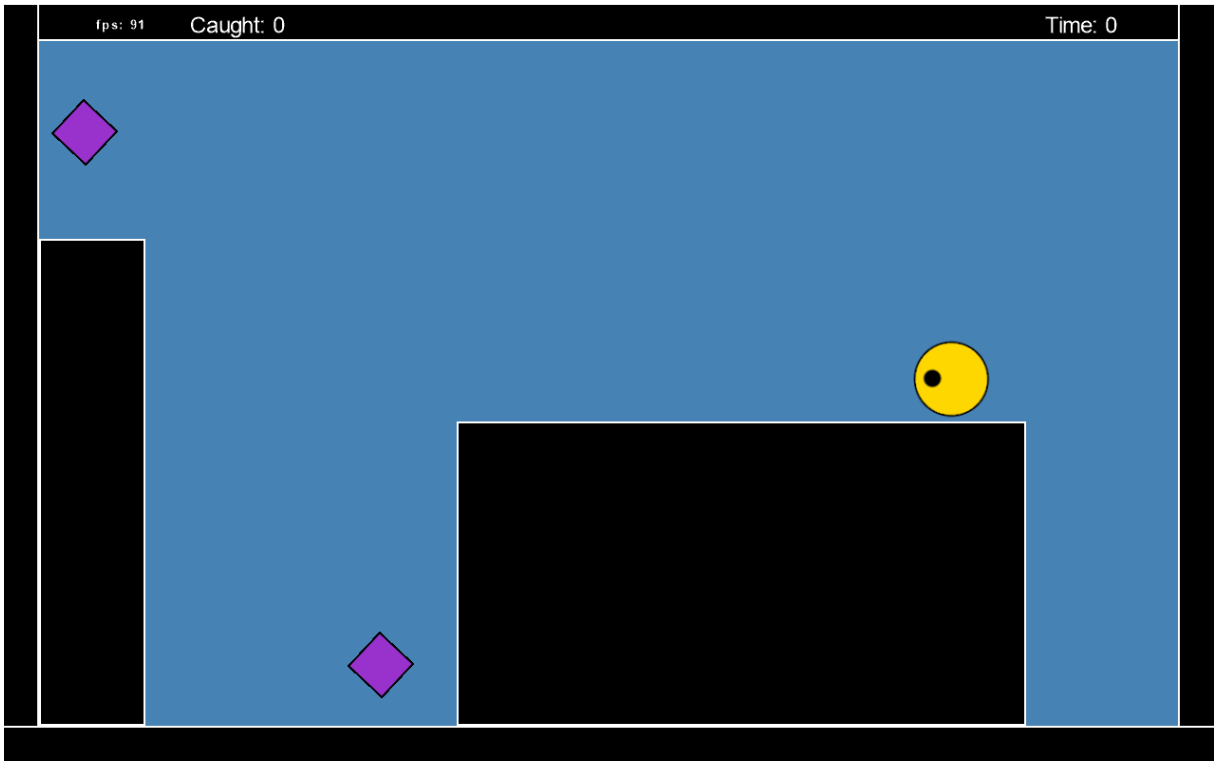


Figure D.6: Geometry Friends 2014 Circle AI Competition – Level 6

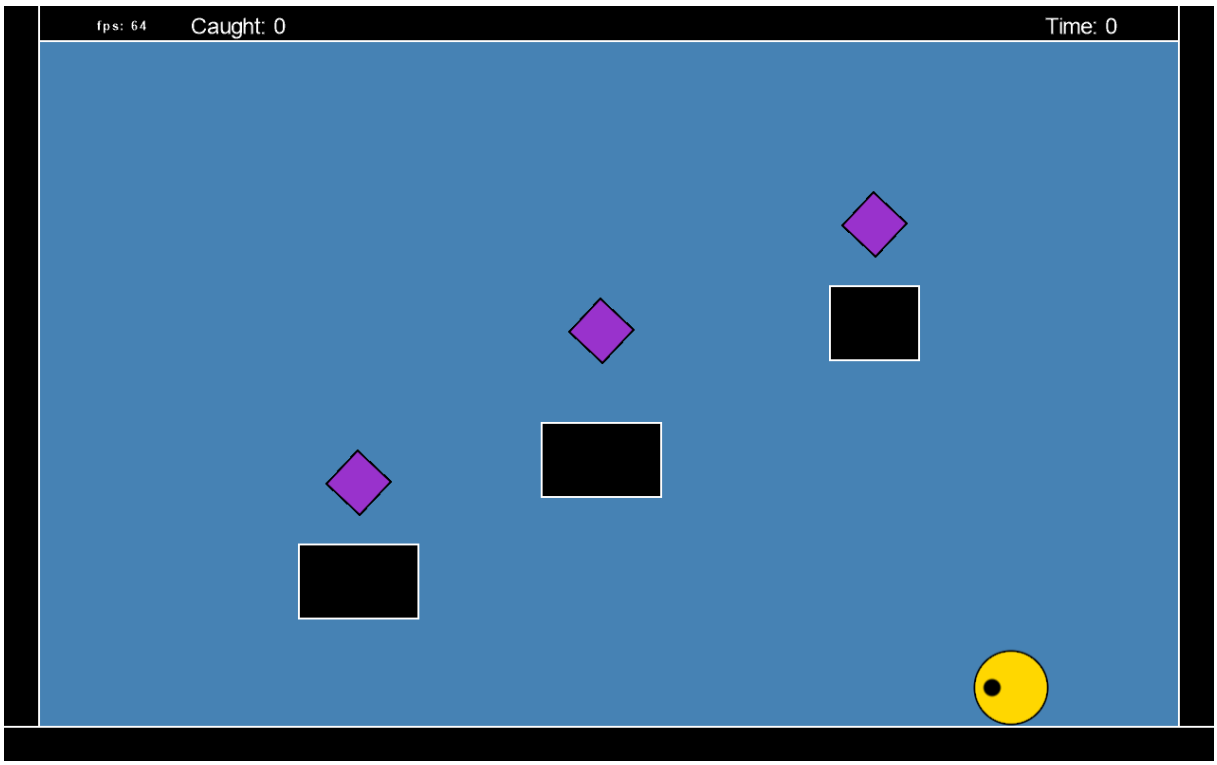


Figure D.7: Geometry Friends 2014 Circle AI Competition – Level 7

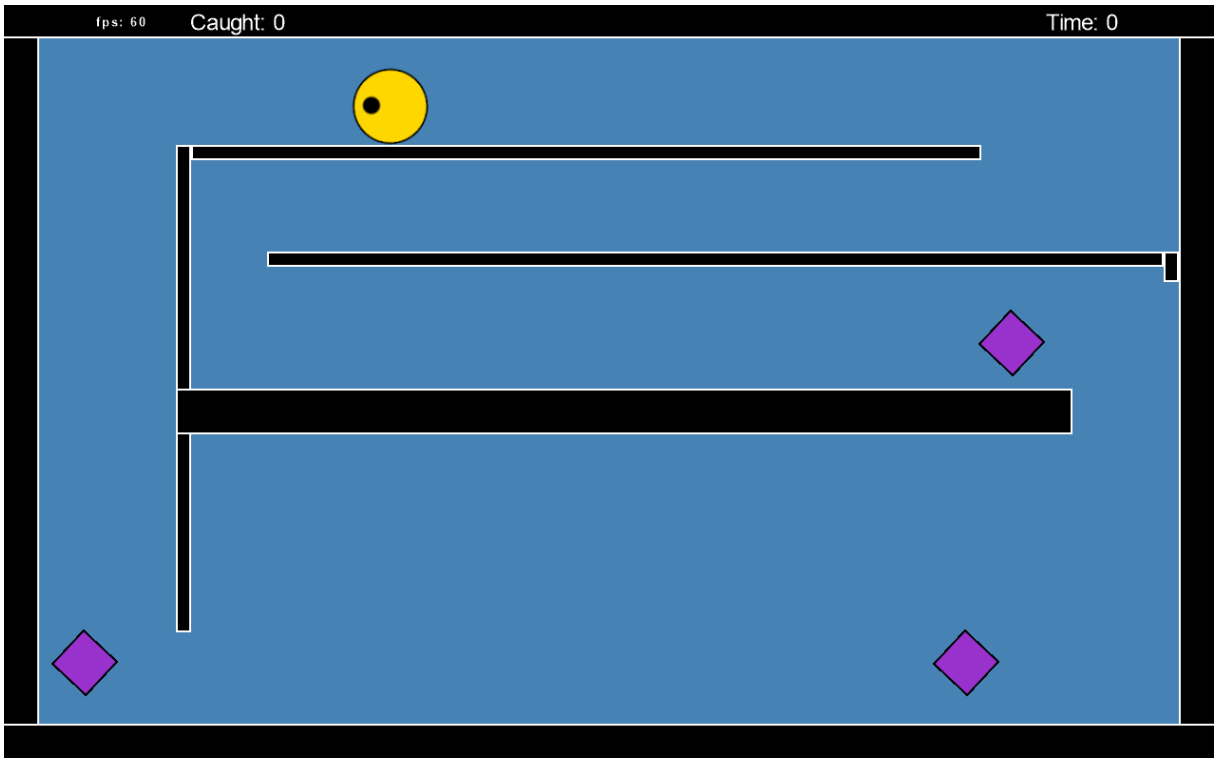


Figure D.8: Geometry Friends 2014 Circle AI Competition – Level 8

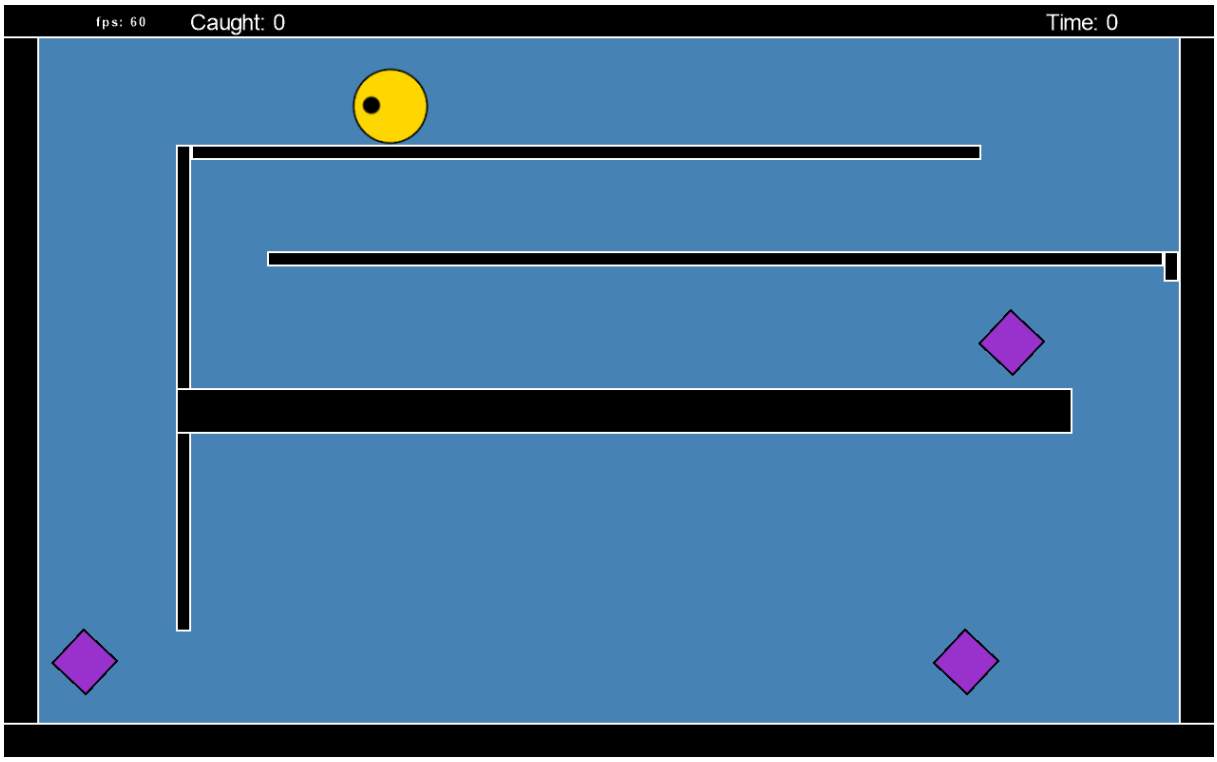


Figure D.9: Geometry Friends 2014 Circle AI Competition – Level 9

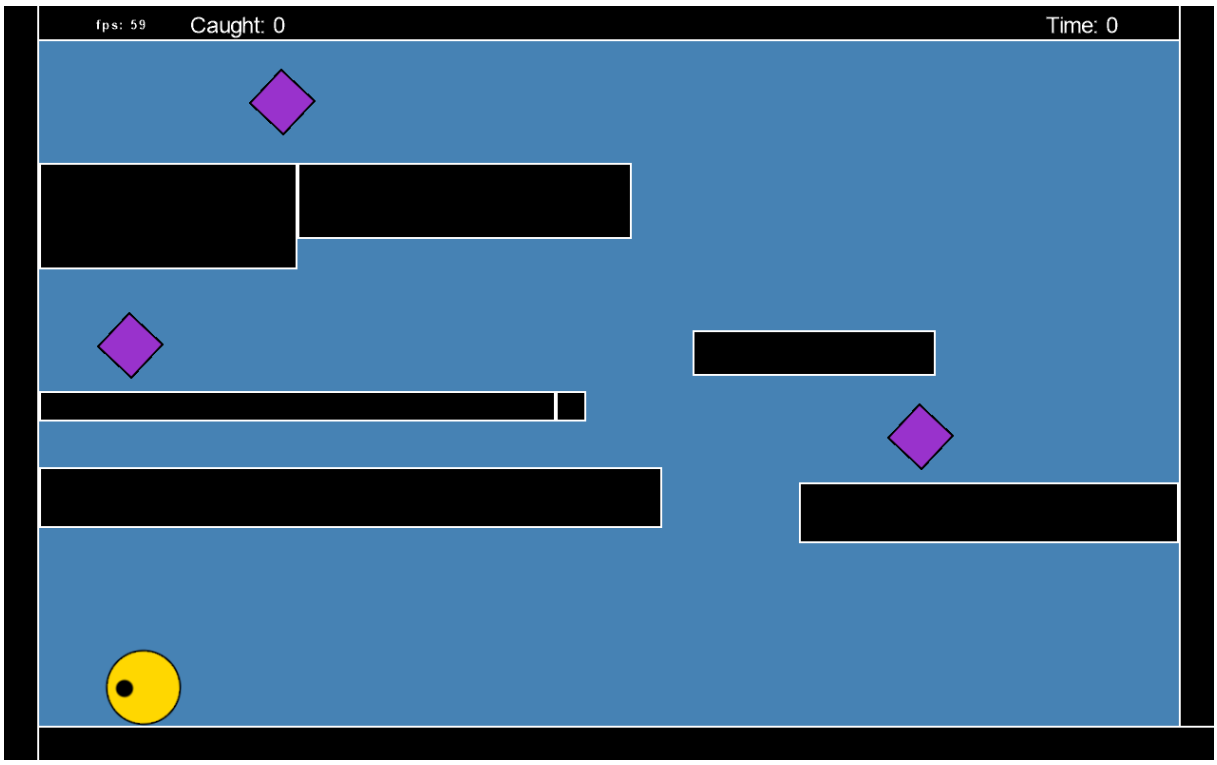


Figure D.10: Geometry Friends 2014 Circle AI Competition – Level 10