Automatic Generation of Social Challenges in Games

DIOGO MENDONÇA

Non Player Characters are an element games can use to provide interesting gameplay scenarios. These scenarios, however, require either being hand written, or a social gameplay system necessitating a large amount of hand crafted rules. By combining a system of social dynamics based on French and Raven's theory of social power with a procedural content generation algorithm, we aimed to create a system capable of autonomously generating such scenarios, foregoing the need for extensive hand made work. In this work, we created an interaction model that implements a simplified version of this theory, a generation algorithm capable of producing scenarios for that model and a game in which to run the model. A generated scenario is a set of starting values that expect the player to follow a specific sequence of actions in order to achieve victory. We performed user studies to assess the effectiveness of our system and concluded that the foundational model and the framework of the generation algorithm did provide the qualities sought, specifically the qualities of being able to generate diverse and expressive scenarios, with flexibility both for customization of the generation and expansion of the algorithm, although there is still room for improvement and some shortcomings, some in terms of the user experience are due to the game itself, and other come from the generating algorithm, such as a lack of control over the generated scenario, and the limited size a scenario can have to be generated in useful time.

CCS Concepts: • **Software and its engineering → Interactive games**; Object oriented architectures; *Software implementation planning*; • **Theory of computation → Randomness, geometry and discrete structures**; • **Human-centered computing** → *User studies*; *Graphical user interfaces*; *Graph drawings*.

ACM Reference Format:
Diogo Mendonça. 2023. Automatic Generation of Social Challenges in Games. 1, 1 (November 2023), 10 pages.

1 INTRODUCTION

Non-Player Characters (NPCs) and the player's interaction with them are a core element of most games. These interactions are usually either pre-defined and fixed or dependent on random chance influenced by the player character's stats. These interactions serve to give the game, and its world, a degree of authenticity, while giving the player some agency in engaging with it.

Recent research [McCoy et al. 2011; Pereira et al. 2016] has aimed to create systems of agents that would dynamically respond to player interaction depending on the social dynamics at play, using as a base some theory from the fields of psychology or sociology. The intent is that such a system would allow for a different kind of player approach to engaging with NPCs, focused on trying to understand them, and their place in society, and how they might get them to cooperate.

Author's address: Diogo Mendonça.

These systems are usually based on sets of rules that define agents, their place in society, what they can and cannot do, and their relationship to the player. One issue with this approach is the need for a large quantity of hand defined rules. The Comme il faut system [McCoy et al. 2011] required "4,900 unique influence rules, nearly 100 microtheories, 30 social exchanges", which severely hinders both scalability and fine tuning.

Procedural Content Generation can be defined as "the algorithmic creation of game content with limited or indirect user input" [Togelius et al. 2011a], and is often used to mitigate issues of scalability [Hendrikx et al. 2013].

This worked aimed to build a procedural generation tool on top of a predefined system for social exchanges between agents, that would allow designers all advantages of social systems of agents, while reducing the quantity of handcrafted work necessary, and furthermore permitting the large scale production of game ready society specifications.

This system was modeled using a theory of social power, power referring to "the probability that one actor within a social relationship will be in a position to carry out his own will despite resistance, regardless of the basis on which this probability rests" [Weber 1978]. The theory we referred to is French and Raven's bases of power[French et al. 1959]. These are coercive power, reward power, legitimate power, referent power, and expert power. For each of these, with the exception of legitimate power, our tool allows for the definition of a relevant social context, and the interactions with the player that the base of power might give rise to.

2 BACKGROUND

In the original paper [French et al. 1959], five bases of power are described, with power being defined "in terms of influence, and influence in terms of psychological change". Psychological change is a process after which the person's system of "behaviour, opinions, attitudes, goals, needs, values, and all other aspects of the person's psychological field" is no longer the same as it was before the process. A base of power is a relationship between a source and a person, through which a psychological change happens to the person. While the number of bases that could be identified is much larger, in the original paper five are identified as being "especially common and important". An important note is that psychological change normally happens due to a combination of different bases, working at different strengths, although the bases can be analysed independently. The rest of this section describes each base of power.

2.1 Reward Power

"Reward power is defined as power whose basis is the ability to reward." Meaning that the strength of this base of power will increase directly with the person's perception of the source's ability to reward them. This ability to reward can be seen either as through giving the person something they value, or by getting rid of something that they wish to see gone. Its value is modified by the perceived

probability of such a reward being granted. This perceived probability is in turn modified by the person's previous history with the source. If the person is usually rewarded by the source, then they will perceive the probability of being rewarded as being higher.

2.2 Coercive Power

Coercive power is similar to reward power, the difference being that while reward power concerns the expected outcome if the person acts as the source wants, coercive power has to do with the expected outcome of not going with the source's wishes. In other words, coercive power comes from the ability to punish. Its strength depends on the intensity of the punishment, modified by the perceived probability of being punished. Much like the perceived probability of being rewarded, the perceived probability of being punished is informed by the person's history with the source. An important distinction between coercive and reward power is that the use of coercive power will reduce a person's goodwill towards the source, while reward power tends to increase it.

2.3 Legitimate Power

Legitimate power stems not from a utilitarian analysis of the predicted outcome but from a set of rules, norms, or values that the person holds, that dictate that the source has the right to hold power over them. This power can have many roots: It can be cultural, stemming from accepted differences of social rank - age, intelligence, cast. "Accepted" is an important aspect. If the social structure is not accepted by a person, no legitimate power will be held over them. Legitimate power can also be delegated. If an agent X has legitimacy in demanding action A from agent Y, an agent Z designated by X will also have the legitimacy to do so.

2.4 Referent Power

Referent power refers to power stemming from a person's feeling of identification with another, with identification being either a feeling of oneness, or a desire to achieve such a state. It is both how much the target likes the source, and how much the targets wants to be like the source. Friendship comes into play here, but also the social status of the source.

2.5 Expert Power

The strength of expert power is how much a person values another's expertise in a relevant area against their own knowledge. This manifests in the source's ability to manipulate the person's set of beliefs, be it at the level of a professor teaching a student or of a local giving directions to a tourist. For such influence to occur, it is necessary both that the person trusts the source to be knowledgeable about the subject, and trusts the source to be telling the truth.

3 RELATED WORK

3.1 Procedural Content Generation in Games

Many PCG methods have been developed for games, with a wide variety of purposes [Shaker et al. 2016]: Search-based approaches search for generated content that best fits a given evaluation function [Cardamone et al. 2011a,b; Togelius et al. 2011b], though their application is mostly academic [Shaker et al. 2016]. There are methods specifically to generate dungeons or labyrinths, using space partitioning [Shaker et al. 2016], digger agents, or cellular automata [Johnson et al. 2010; Van Der Linden et al. 2013]. Some examples of different methods to do PCG:

- **Grammar Based Approaches**: As non-deterministic specifications of systems, grammars can be applied to generate content. One example would be the work of Shaker et. al [Shaker et al. 2012], who used grammatical evolution to generate levels for the game Super Mario Bros.
- **Planning Approaches**: Planning algorithms are searches for the satisfaction of a goal through a sequence of actions. This sequence of actions can then be interpreted as a story. The two main ways of doing this story generation are through plan-space search, in which the algorithm explores possible plans for a path to the goal, and state-space search, in which the algorithm builds up the plan by adding new actions, until an acceptable state is reached. As an example of planning based generation, Riedl et al. [Riedl and Young 2010] developed a system to generate stories using an algorithm, the Intent-based Partial Order Causal Link (IPOCL) planner.
- **Search Approaches**: PCG can be done through a search approach, in which case the system is decomposed in three components [Shaker et al. 2016]: a search algorithm, which will seek out the best solution to a problem; a content representation, which represents the artefact to generate, be it an agent, a world, a dungeon, etc; and evaluation functions, which attribute to any given generated artefact a numeric value corresponding to its quality. A commonly used type of search algorithms are evolutionary ones [Shaker et al. 2016] which can be described as ones that start with a stochastic population, iteratively evaluate its individuals, pick the best ones, and generate a new population based on that best set [Eiben et al. 2003]. The process is repeated a set number of times or until an individual with high enough quality is found. Other than evolutionary algorithms, others can be used for the search. In small search spaces, it is feasible to do an exhaustive search that tests each possible solution. When finding good solutions is relatively easy and diversity is more important than optimization, a random search might be enough.
- **Machine Learning**: There are several approaches to applying machine learning to PCG problems: graph-based representation allows for a generic solution that is not bound in shape or size, while other representations such as matrix, sequence, or grid introduce explicit constraints. [Summerville et al. 2018] As for the training, while there are variations in terms of space and time efficiency, in general, the approaches follow the principle of iterating over a corpus adjusting internal values until a specified limit.

  The generation of content using machine learning presents some specific problems: It's difficult to guarantee that the generator would only generate playable / possible levels, with implementations usually applying designer-specified constraints [Snodgrass and Ontanón 2016; Summerville et al. 2018] to ensure playability. Another issue has to do with the

training data. For machine learning approaches to work effectively, they require a large sample to train from. The larger the data set, the better the results [Halevy et al. 2009]. However, for games it's hard to build a large corpus. Despite there being a large number of games, they rarely share data structures that can easily be packed together, nor do they share the same semantics, even when the data structure is similar [Summerville et al. 2018]. As a result, data sets are usually composed of data from within a single series [Summerville and Mateas 2015] or an individual game [Guzdial and Riedl 2016]. While this does achieve some results in replicating an existing game, it quite limits the generation of new ones.
Another way of using machine learning to generate content is through reinforcement learning [Khalifa et al. 2020]. Reinforcement learning is typically used to train agents to play a game [Justesen et al. 2019], but it can be adapted for generating content if the generation is seen as itself a game [Khalifa et al. 2020]. As this approach does not require a training set, it bypasses one of the biggest weaknesses of machine learning PCG, the lack of large, unified corpora to train from. When compared to search-based approaches, the biggest advantage is time. Although it takes time to train, after it has been trained it generates content quickly, while search based algorithms will have to perform the entire search for every generated artifact.

3.2 Social Gameplay

There has been some amount of research into creating social gameplay, such as the aforementioned Comme il Faut[McCoy et al. 2011] system, which provides a "playable model of social interaction where the author provides reusable and recombinable representations of social norms and social interactions". It does this by going through a process of desire formation, intent selection, performance of social exchange, and social fallout. This is, as mentioned before, based on rules, which can number in the thousands.

3.2.1 SAPIENT system. Pereira developed a system named SAPIENT [Pereira et al. 2016], integrating social power dynamics into a cognitive agent architecture, using as a foundation French and Raven's bases of power, the very same theory we used for our project. This system was then used in the context of a virtual environment to generate character behaviour.
The system models interactions in which an Agent wants a Target to exhibit a Behaviour, with the binary outcome of whether such a Behaviour is performed or not. To make that decision, a quantitative measure of the influence of each base of power is calculated, taking into account the Target's biases and beliefs. These are then added together and weighted against a value intrinsic to the Behaviour itself, resulting in a numeric value. If this value is positive, the Behaviour will be performed. If it is not, then the Behaviour will not be performed. The quantitative measure of the influence of each base of power is calculated differently for each base. There are also differences in how each base affects the dynamic between agents. As an example, if a reward influence causes the desired Behaviour, the Target will expect the Agent to reward them.

We'll succinctly go over each base of power and how it is handled:

- **Reward power** is calculated from the sum of all actions the actor can reward the target with. Each actions is valued by the agent, multiplied by their bias, and multiplied by how much the target believes the agent will reward them. On a successful interaction an expectation of the agent rewarding the target is created. Its fulfillment, or lack thereof, will influence the target's belief of how likely the actor is to perform rewarding actions. Its fulfillment will also increase the target's opinion of the actor.
- **Coercive power** is similar to reward power. It is calculated from the sum of all actions the actor can punish the target with. Each actions is valued by the agent, multiplied by their bias, and multiplied by how much the target believes the agent will punish them. On a successful interaction an expectation of the agent not punishing the target is created. Its fulfillment, or lack thereof, will influence the target's belief of how likely the actor is to perform punishing actions. Regardless of its success, coercive power causes a decrease in the target's liking of the actor.
- **Legitimate power** works in the context of social groups. For social power to exist, both agent and target must exist within a group relevant to the behaviour in question. For the groups they both belong to, the importance of the group to the target is multiplied by the target's bias and how much the agent perceives the group to conform to norms.
- **Referent power** is comes from two sources: how much the target likes the actor; and the social status the target perceives the actor to have.
- **Expert power** depends on the actor's skill, and how much the target trusts it. It's a simple multiplication of the skill difference between them by the bias and the trust placed by the target on the actor.

Our own solution was based on the SAPIENT system, though heavily simplified to allow for a both procedural generation and the gameplay that feels natural to the player.

4 INTERACTION MODEL

The interaction model we used shares some similarities with the SAPIENT system, although somewhat simplified. An influence attempt $I(P, T, B)$ is composed of the player P, a target T, and an expected behaviour B. Its outcome is either the expected behaviour is performed, or it is not.

$$I(P, T, B) = CombinedPower(P, T, B) - Resistance(T, B) > 0$$

$Resistance(T, B)$ refers to how much the target does not want to perform the behaviour, and is defined in the society's specification. $CombinedPower(A, T, B)$ refers to the total sum of the influence of all bases of power considered. As in the SAPIENT system, ISP refers to the set of considered bases.

$$CombinedPower(P, T, B) = \sum_{p \in ISP} BasePower_p(T, P, B)$$

Each base is additive, meaning multiple uses of any base will add to the ones already used. For example, giving an agent both a fork and a knife will yield the power of the two individuals combined.

Giving multiple of the same item, however, will not increase the power. Likewise for threats, appeals to skills or social groups.

4.1 Reward Power

As a simplification, we consider only the value of an offer o, which is given before the interaction. $RewardBias_T$ represents the target's inclination to accede to reward power, and is defined in the society's specification. $Value_{T,o}$ refers to how much the target values what is being offered, and is likewise defined in the society's specification.

$$BasePower_{Reward}(T, P, B) = Value_{T,o} * RewardBias_T$$

Should the target accept the reward, it will result in an increase in its liking of the player, represented by a numeric value $RelationT, P$, which will come into play when calculating referent power. The increment is half the valuation the agent has of the item.

4.2 Coercive Power

As a simplification, we consider only the value of a threat t, assuming that the target fully believes in its fulfillment. $CoerciveBias_T$ represents the target's inclination to accede to coercive power, and $Value_{T,t}$ refers to how much the target wants to avoid what is being threatened. Both values are defined in the society's specification.

$$BasePower_{Coercive}(T, P, B) = Value_{T,t} * CoerciveBias_T$$

A significant difference in relation to Reward Power is that the player electing to try to coerce an agent (regardless of the success of the cohesion) will damage the target's opinion of the player, represented by a numeric value $RelationT, P$, which will come into play when calculating referent power. The decrement is half the valuation the agent has of the threat.

4.3 Referent Power

As a simplification, we consider referent power to be solely based on how much the target likes the player avatar. $ReferentBias_T$ represents the target's inclination to accede to referent power, and $RelationT, P$ is a numeric measure of the target's opinion of the player. They are both initially defined in the society's specification, however $RelationT, P$ is a dynamic value that evolves with the player's actions.

$$BasePower_{Referent}(T, P, B) = Relation_{T,P} * ReferentBias_T$$

4.4 Legitimate Power

To calculate legitimate power we will make use of the concept of social groups. Within such groups, agents can have distinct roles, which establish a hierarchy between agents. When an interaction based on legitimate power happens, the player will be leveraging the influence of a group g they belong to to influence the target. $LegitimateBias_T$ represents the target's inclination to accede to legitimate power. It, along with which agents belong to which groups and have which roles, is defined in the society's specification.

$$RoleD_{P,T,g} = RoleValue_{P,g} - RoleValue_{T,g}$$

$$BasePower_{Legitimate}(T, P, B) = RoleD_{P,T,g} * LegitimateBias_T$$

$RoleValue_{x,g}$ is a measure of the power associated with the role of x within group g. In other words, it represents how much power the group's internal hierarchy confers to members of that role. $RoleD_{P,T,g}$ represents the difference in those powers between player and target, and thus how much power the group's structure confers to the player over the target.

As a simplification, we will not be taking into account how relevant a given social group might be for an interaction or how much value an agent places in a given group.

It should be noted that while the model does implement legitimate power, no nodes that make use of it were implemented. Thus, as it stands legitimate power does not come into play at all in any society generated by our generating algorithm, though in one edited by hand to include social groups it would be able to. Furthermore, it should be of relative ease to posterior expand our generator with nodes that do make use of this base of power, in future work.

4.5 Expert Power

To calculate expert power we will make use of a similar strategy to what we used for legitimate power. We will define skills, within which agents can have distinct levels, which establish a hierarchy between agents. When an interaction based on expert power happens, the player will be leveraging the difference in a skill s to exert power over the target. $ExpertBias_T$ represents the target's inclination to accede to expert power. It, along with which agents have which skills, is defined in the society's specification.

$$SkillDifference_{P,T,s} = SkillValue_{P,s} - SkillValue_{T,s}$$

$$BasePower_{Expert}(T, P, B) = SkillDifference_{P,T,s} * ExpertBias_T$$

$SkillValue_{x,g}$ Is a measure of the skill level associated with the role of x in the skill s. In other words, it represents how good at the skill an agent is. $SkillDifference_{P,T,s}$ represents the difference in those levels between player and target, which will influence how much power will be conferred to the player.

5 SOLUTION

We'll start by describing the game in which the model is inserted, to understand the functionality it must provide. The intent is to have a system that can be plugged into several different types of games, so the game specification should not matter much to the generation and society processes. However, there's one layer of the generation algorithm that must be tailored to game it is intended to run on, and as such, it will be easier to go over the generation algorithm if we already understand how the game works.

The entire implementation of the project is available online here: https://github.com/GAIPS/SocialChallengesPCG

5.1 The Game

The game has 3 elements:

(1) A player controlled character;
(2) A number of agents;
(3) A vending machine.

The player character can move around and interact with other elements. It also has a list of items it possess, a list of incriminating

information it knows, and a skill value for each of the player's skills. NPC agents have a list of behaviours they can perform, and the vending machine provides items, free of charge, to the player. Finally, during an interaction with an NPC agent, the player is given some options:

- **Talk**: Offers some dialogue lines that give the player some information about the agent's values. More specifically, it checks for all the actions available to the player, and in case there is at least one that would give power to the player, a clue will be given.
- **Ask**: Lets the player ask the agent to perform one of its assigned behaviours, the success of which depends on the power the player has over the agents.
- **Act**: Allows the player to perform actions which will affect the power the player has over the agent.

The typical game begins with the player going to the vending machine to get a free item, so they have something to start gaining power over, and influencing, NPC agents. From there, the player will interact with different agents, performing actions that raise their power, and influencing behaviour that give them tools to get more power (be it by obtaining new items, information, or skills). At a certain point, the player will be able to influence the victory behaviour of one of the agents, thus winning the game.

Each influenced behaviour has consequences which give origin to four different actions that can occur:

(1) If the player is given an item, they can give it to an agent, obtaining reward power, and a raise in the relationship value;
(2) If the player has a skill raised, they can point it out to an agent, obtaining expert power;
(3) If the player is given incriminating information about an agent, and they are interacting with the concerned agent, they can threaten to reveal that information, obtaining coercive power, and a drop in the relationship value;
(4) If the player is given incriminating information about an agent, and they are interacting with a different agent, they can share that information, obtaining reward power with the agent they are interacting with, while lowering the relationship value with the agent the information is about;

5.2 The Society

With the game described, we can now describe a society. A society is comprised of two elements: a set of agents; and a set of starting items.

Each agent has the following fields:

(1) Biases;
(2) Social Groups;
(3) Skills;
(4) Relationship Value;
(5) Valuations;
(6) Behaviours;
(7) Current vales.

At the start, the game populates its world with the agents defined in the loaded society, and fills the vending machine with the starting items.

5.2.1 Biases. There is one bias for each base of power, represented by a floating point value. This represents how much the agent tends to concede to one type of power over another.

5.2.2 Social Groups. Each agent carries an array of the groups they are inserted in. Each entry contains two fields, both strings: One identifying the social group the agent belongs to, the other identifying the role the agent has within it.

A separate array is stored, in the society itself, containing all groups that exist, and all of their roles, and more importantly, a numerical value that represents how much power each role has within the group.

5.2.3 Skills. Each agent has an array of skills, representing how skilled they are. Each entry has two fields. One is a string that identifies the skill, the other is a numeric value that identifies how skilled the agent is at it. Non included skills are assumed to be 0.

5.2.4 Relationship Value. It represents how much the agent likes the player. This value is a simple floating point number, and it starts at zero. It is changed as the player interacts with the society.

5.2.5 Valuations. This represents how much the agent values something that the player gives them, or threatens them with. It's an array in which each entry contains a string identifying the target of the valuation, be it an object, threat or piece of information, and a numeric value quantifying how much value the agent places on it.

5.2.6 Behaviours. Behaviours are the actions players can request of agents. They include both the resistance the agent has to them, and the consequences of successfully influencing the behaviour.

5.2.7 Current Values. These represent the power the player has over the agent at any given point of time. They are all floating point values, and there is one for each base of power. They are initialised to zero, but as the player acts towards the agents, they are raised or lowered.

5.3 The Interaction Engine

Our interaction engine comes into play at two different instances:

When acting the player is acting, the interaction engine is called to update the agent's current values, with accordance to our interaction model (Section 4).

When the player is asking, the interaction engine sums all of an agent's current values, and if the result is larger than the agent's resistance to the behaviour, then the interaction is successful. Otherwise, it is unsuccessful. Successful interactions result in the behaviour being performed, while unsuccessful interactions do not result in any consequence.

Regardless of success, after asking, the current values for both the reward and coercive bases of power are reset to zero. We consider the "bribe" or "threat" to have been expended.

5.4 The Generation Algorithm

The purpose of the generation algorithm is to generate a society which presents a problem that can be solved, in the form of agents with fields set in a way to allow an interesting (and possible) path for the player to reach the goal behaviour.

There are fields which can be zero at the start of the game, and thus need not be generated, both for the sake of simplicity, and because it makes sense within the context of the game. These are the relationship and current values, as we assume a scenario in which the player character has no past history with the agents, and thus, no power over them, nor do they have an opinion of the player. The rest of the values need to be generated.

The generation algorithm has three phases, all independent from each other, in the sense that they can be implemented in different ways without affecting each other, as long as the output of each phase remains within the necessary constraints.

(1) The **Building Phase**: A graph is constructed representing the sequence of actions the player must take to reach the behaviour goal, which is the root of the graph.
(2) The **Spawning Phase**: Each node in the graph is given meaning in the context of our game. This means that a node that represents a gift behaviour will generate the specific item it represent, for example;
(3) The **Validation Phase**: Each node in the graph is traversed in from the starting nodes to the end node, simulating the sequence of actions the actual player would take. At each step, society values are changed to ensure it is valid.

5.4.1 The Node. Before proceeding, it is important to specify what a node in our graph represents. If we think about how the player interacts with the society, a pattern emerges. The player successfully influences a behaviour, that behaviour results in a new state of the world, in which the player can perform a set of actions to successfully influence a different behaviour. For example: The player can successfully influence an agent John into giving them a hat. The player can then give that hat to another agent, Joe, to get enough power to influence some behaviour of Joe's. We can represent this graphically by having each node be a behaviour, and each (directed) edge the actions required to influence it. The edge is not explicitly defined, being instead inferred from the two nodes it connects.

There are several types of nodes. Note that none of these nodes is forcibly required. Any of them can be replaced by different ones that serve a similar purpose. We can also have several nodes using the same base of power. Reward power, for example, is used by the Free Item Node, Gift Node and Secret Node alike.

- **Victory Node** It represents the behaviour the player must influence in order to win.
- **Free Item Node** It represents an item that should appear in the vending machine in the game. Functionally, it works the same as a gift node, as the actions emerging from it also rely on reward power.
- **Gift Node** It represents a behaviour which gives the player the associated item. Any successor node will then make use of the reward power granted by this node to have its behaviour achieved.
- **Secret Node** This node has one piece of incriminating information, the agent to whom that information refers, a resistance and a agent that discloses that information, which may or may not be the same as the one the information concerns. The node represents a behaviour which gives the player the associated information.
- **Skill Node** It represents a behaviour which raises the player's skill by the associated increment. Any successor node will then make use of the expert power granted by this node to have its behaviour achieved.

5.4.2 The Building Phase. In this phase, a graph is constructed to represent the sequence of actions the player must perform to successfully complete the game. At the point this phase ends, that sequence is completely meaningless in terms of content, having only the information of the type of node, with the graph representing merely an abstract connection between undefined actions. In our implementation, we used an abstraction to allow easy swapping of algorithms. We will now discuss the algorithms experimented with, but before, some terms need to be clarified.

Because different algorithms can have different interpretations of what a parent is, it is necessary to distinguish between parental relationships, and successor/prerequisite relationships. A node's **parent** refers to its relation in the context of the process of building the graph, while a **successor** means a node which behaviour will be performed after the behaviour of the **prerequisite** node. To illustrate the importance of this distinction, a graph building algorithm can have its root be the Victory Node, in which case the parent of a node will be its successor. On the other hand, an algorithm which uses a Free Item Node as the root for its graph will have each node be prerequisite of its children.

The final note about nodes before we get into the algorithms is how they are categorized.

Nodes can be categorized into three different categories: **Starting Nodes**, **Middle Nodes**, and **Ending Nodes**. This is in reference to their successor/prerequisite relationships. **Starting Nodes** have successors, but not prerequisites (an example would be the Free Item Node); **Ending Nodes** have prerequisites, but not successors (the immediate example would be the Victory Node); **Middle Nodes** have both prerequisites and successors (and in our implementation correspond to the remaining nodes: Gift, Secret and Skill nodes).

. Tree Generation Algorithm

This is the first algorithm tried. It starts at an Ending node, randomly generates a number of children for it, which can be either Middle Nodes or Starting Nodes, then repeats the process for every child that is a Middle Node until a maximum depth is reached. Afterwards, for every leaf node that is not a Starting Node, a Starting Node is added as its child. This ensures all leaf nodes are Starting Nodes, and the player is expected to perform the associated behaviours and then move up the graph until reaching the Ending Node. When interpreting a graph generated by this algorithm in further steps of the overall process, each parent relation will be interpreted as a successor, and each child relation as a prerequisite.

This approach has the advantage of providing for a simple and quick algorithm, but results in a society that has a lot of starting items, making the actions available to the player that much wider, which could result in some confusion. In addition, each behaviour can only be used for a singular successor, which is an unnecessary limitation that might trivialize the challenge of figuring out how to most effectively use the actions the game gives you.

. **Metalike Algorithm**
This algorithm is inspired by the Metazelda project https://github.com/tcoxon/metazelda. Similarly to the Tree Generation Algorithm, it creates a tree by randomly generating children until a max depth is reached, with the difference of using a Starting Node as root. Afterwards, a random childless node is chosen to be the Ending Node. Recursively, the ending node and all its ancestry is marked as *End-able*. Then, for each childless and non-*End-able* node, a random *End-able* node that is not an ascendant of the childless node is selected, and becomes the child of the childless node. The formerly childless node and all its parents are marked as *End-able*. After this process is repeated for all non-*End-able* leaf nodes the graph is done. This ensures that:

- All nodes are descendant of the starting node;
- All nodes are ascendant of the ending node;
- There are no loops in the graph.

This algorithm is more complex, and more computationally expensive, than the Tree Generation Algorithm, but it has the advantage of producing a society that not only has only one singular starting item, but contains actions with multiple successors, making each interaction more valuable, and with the potential to be more interesting.

When interpreting a graph generated by this algorithm in further steps of the overall process, each parent relation will be interpreted as a prerequisite, and each child relation as a successor. The opposite of how it was handled with the Tree Generation Algorithm.

5.5 The Spawning Phase

In this phase, each node in the graph is given meaning in the context of our game. In turn, each node is visited, and the required parameters set. What parameters are needed will vary between the different types of nodes, and is described on section (5.4.1), for each type of node. The agent associated with the node is randomly set, and a behaviour is constructed, and added to the agent in the society class.

5.5.1 The Validation Phase. In this phase, the graph is traversed, simulating a player's sequence of actions, and the society's fields are changed to ensure the game is winnable.

The algorithm begins by putting all of the Starting Nodes in a FIFO queue. It then takes the first node from the queue. It checks whether the node's resistance is surmountable or not. In the case of the Starting Nodes, which by nature have no resistance, this check always passes; For the rest of the nodes, they'll have to calculate how much power each of their prerequisites grants them, to which is added the referent power, which despite having no node directly associated, is influenced by them, as both reward and coercive nodes have the side effect of changing the player's relationships. If the combined power is larger than the node's resistance it passes, and the node's successors are then added to the queue. Otherwise, a random field associated with the node is raised, and the check is run again. An example of a risible field would be, for a node associated with agent John, which has a prerequisite Gift Node with an ice cream, the valuation John has of ice cream.

There are some variables that are more permanent that the immediate next interaction. For example, a skill the player receives will remain with them, and changes to relationships are not reset after each interaction. It's required to store these in a permanent structure that remains accessible throughout the validation phase. This is where the Run-time Model comes in. It stores player skills and relationships, and is updated after each node is validated. It is referenced by the nodes when checking their validity, and ensures that the validation is accurate to the parameters a player following the same sequence of actions would have at any given time.

5.6 Game Integration

When a scenario is loaded, a SocietyIOManager obhect comes into play. This class reads the society file, generates a Society class that holds the file's information, and will be responsible for any future interaction with it.

An AgentSpawner reads the agent list from the society, and spawns in the game an avatar for each of them. Each avatar has a InWorldAgent object in them, which simply carries an agent id so that the model can know which agent the player is interacting with.

The player has a controllable avatar, carrying a PlayerController, and all it does is allow movement, and start interactions.

The vending machine opens up a UI screen whenever it is interacted with, displaying a button for each of the starting items. On the pressing of a button, the player is given the associated item. Items, threats and skills are stored in the PlayerController.

On interacting with an agent, a dialogue interface opens up. It displays the agent's current values, name, and a greeting. Four options are given to the player:

- Talk
- Ask
- Act
- Leave

Leaving causes the dialogue interface to close.

5.6.1 Talking. The objective of talking with an agent is to obtain information that might tell us what the agent will find valuable. The list of actions the player has available for interaction with that agent is combed through. calculating how much each is worth, and for each that is worth more than 0.1 power, a dialogue line is added to the agent's pool that strongly suggest the action. Each time the player chooses to talk with an agent, one of the lines in this pool is displayed.

If it so happens that the player does not have any powerful action available to them, the agent will then simply remake "Nice weather we're having".

5.6.2 Asking. When asking an agent, a new interface appears, showing the player the list of all the behaviours that agent can perform. The player picks one from the list, and the InteractionController class is called, which uses the interaction model described in 4 to determine whether or not that interaction is successful. If it is not, the player is told so. If it is, then the behaviour happens, triggering the consequences delineated in its Consequence class.

5.6.3 Acting. When acting, a list of available actions is constructed.

- For each item in the inventory, a *Give Item* action is added

- For each piece of information in the inventory, if the information is about the agent the player is currently acting on, a *Threaten With Information* action is added.
- For each piece of information in the inventory, if the information is **not** about the agent the player is currently acting on, a *Share Information* action is added.
- For each skill the player has, a *Show Off Skill* action is added.

This list is then displayed to the player, who can pick any action to perform, and their consequences, as describe previously, happen.

6 USER TESTING PROCEDURE

Three different scenarios where prepared, all using the Metalike algorithm.

The first scenario had a maximum depth of 2, and a maximum children number of 2. This created a very simple graph with only 3 nodes: Free Item Node, a Gift Node, and a Victory Node.

The second scenario had a maximum depth of 2, and a maximum children number of 3. This graph was still very simple, having 4 nodes, but it introduced a Secret Node and a Skill Node.

The final scenario had a maximum depth of 3, and a maximum children number of 2. It was the most complex graph, with 6 nodes, 3 of them being Skill Nodes, and the other a Secret Node that is both used as a threat and a gift.

The players are asked to play these three scenarios in succession, advancing whenever they beat the scenario. As it might be possible reduce the relation value with an agent too much, a button was added to the user interface allowing to reset a scenario to its initial state. Player actions in-game are recorded (including those of failed attempts). After playing, they are asked a series of questions. These are meant to give us a better understanding of what the player felt playing the game, and how where they thinking while playing it, in the hopes of it giving us a better understanding of how well our system was working, its shortcomings, and how it could eventually be improved.

7 RESULTS

There are two categories of results that will be discussed:

(1) The direct results obtained by experimenting our generator, both with different parameters and algorithms, and by maintaining the parameters constant, but changing the seeds used.
(2) The results obtained through user testing.

We will start with discussing the direct results.

7.1 Direct Results

The algorithms used allow for the creation of diverse graphs, which in turn create social situations of different complexity to solve.

The biggest factor of variance is the algorithm used. We experimented with merely two, but even by comparing the graphs generated by them it's quite clear just how different they are.

For the Tree Generation algorithm, it does not have much diversity in terms of shape. All graphs end up looking like a tree, as the name indicates. The contents of the nodes and the size of the problem varies, but most graphs still look very similar.

The Metalike algorithm, on the other hand, offers much more variability in terms of the shape of the generated graphs. It can range from separate, clear paths to victory, to inter-crossing strands of different paths, to an inscrutable confusion of nodes and edges.

As for the parameters of the algorithms, the most immediately consequential parameters of the algorithm are the maximum graph depth and the maximum number of children per node. These two combined determine both shape and size of the graph. Graphs with low values will present simple problems, whose solution can be reached in a handful of steps, while large values, on the other hand, will create complex webs of nodes that are difficult to parse visually and even more difficult to solve in-game. Small graphs would thus be more suited for initial learning levels, and larger graphs to later more advanced levels.

These parameters are limited by how they make the graph scale, however. For both Metalike and Tree Generation algorithms, the maximum number of nodes that a tree can have increases exponentially with the increase of maximum children nodes and maximum depth. From our experiments, a maximum depth of five, with the maximum number of children set to four already almost always results in graphs too complex to be viable, at least in the current implementation of the game.

The stochastic nature of the generation allows for a good measure of diversity between graphs of the same parameters, but this is a double-edged sword that comes at the cost of control-ability. For example, as only the maximum number of children is a parameter, with the minimum always being one, it is very possible for a graph with a large maximum children number to simply be a straight line of single children, which could go against the wished of a designer who has aiming for a complex scenario.

The size of the graph also affects how fast it is generated. For small parameters, the time it takes to generate a graph is negligible. It's a fraction of a second that could be hidden in the loading of a scene without causing any noticeable change in loading times. For larger graphs, it might take a few seconds to generate, while large graphs will take unreasonable amounts of time. If a game's intention was to have societies be generated at run-time, then large graphs would not be feasible. These would only be acceptable to use if the society was to be generated beforehand, by the designers. Even then, from our measurements, the generation algorithm is quick to reach a size where it requires multiple hours to generate a scenario.

From the parameters that can be set, the number of agents is the one of the least consequence. In terms of the shape of the graph, the number of agents is not even a factor, as nodes only make use of such information after the building phase.

What the number of agents does influence is the amount of interactions that are made with each. Having many agents for few nodes will make it so there are some agents with whom the player does not even need to interact with, while having a few agents for many nodes will make the player interact with each agent several times.

7.2 User Testing Results

Players' ages were within the 18-25 range, with 73% being male and the rest female. Most of them play games regularly, with 33% playing daily and 47% playing a few times per week.

Starting with the questions present on the form, it is clear that players had no trouble understanding how to control their avatar, and the games mechanics, on the surface were generally understood. However, players did not feel like they had mastered them, and feelings of being good at the game were not very strong. There was some confusion associated with the mechanics, particularly, per their written feedback, uncertainty related as to whether or not any given action would be enough to surpass the behaviour's resistance.

One other interesting takeaway from the questionnaire was that players felt the game was too abstract in its presentation. There are two factors that were explicitly claimed by the players to be causes for this, thought it is unlikely they are the only ones at play. These factors are, according to the players who left feedback: The game displaying power numbers, aggravated by the fact that they are not at all meaningful for the player; and the lack of perceived rationality in the agent's decisions. This last one is likely due to the disconnect there is between different actions and behaviours. All this also contributed to reduce the feeling of freedom of choice players had and led to a lack of a feeling of immersion, a lack of a feeling of consequence, that the player's actions aren't really doing anything that chances the society itself. To this last point it probably also contributes the fact that there really isn't much difference in how the society is before and after any given action, besides the relationship between the agent and the player, which even then is but a mere number.

All in all, the general conclusion is that the game feels too abstract, too about the numbers, while everything else is mere set dressing.

In terms of the paths player had available, most of them felt like they did not have much choice in terms of what to do to achieve victory, despite most of them beating the scenarios through sequences of actions that were not the ones outlined in the scenario graph (though on this point most players reported that they did indeed believe they had reached the intended solution).

One of the observations from the testing data was that the current interaction model allows for brute-forcing a solution, meaning, repeating the same failed strategy over and over again until it succeeds. This is possible due to how the relationship value works. Every time an item is given, the player gets reward power and the relationship value increases. When the player tries to influence a behaviour and fails, the reward power is reset to 0, to avoid this very problem. The relationship value, however, isn't. This means the player can quite simply repeat a gift several times, slowly increasing the relationship value, until the referent power is high enough to make up for any other missing prerequisite.

As for how players were thinking and planning during the game, most reported being unsure of how well they were doing at the game in the middle of it, which is further supported by observation of the play data, which tells us that players were resetting the scenarios even when they were not in a situation worse than the initial one. There was no strong feeling of deliberation in their actions, and the number of players who reported acting based on what they have only was identical to the number of player who reported acting based on the goal they were trying to achieve.

Most players reported having enough information to make informed decisions, but it is clear from analysing their play data that there was still some confusion. According to their written feedback, and as seen from the play data, most players were trying whatever action was available, effectively searching through the available action until one succeeded. This approach worked for our simple scenarios, but would make more complex scenarios very hard to deal with.

One difficulty in forming plans might come from the fact that players only have information about the behaviours agents have and the items/threats/skills the player has. This correlates to another issue players had: Most tried to talk with agents before they even had items, which, due to how talking was implemented, meant agents had nothing to tell them.

Despite some lack of clarity, almost all players thought the game was easy. The scenarios were too simple for the difficulties they were having in understanding the scenarios and making plans to manifest themselves in problematic ways. Some players had difficulty in the first scenario, due to misconceptions they had about the rules, or some concept or the other being unclear. As soon as they figured it out and moved on to the second scenario they stopped having difficulty.

Most players could understand the difference between scenarios, which is a positive evaluation of the generator's expressivity. As for diversity, there was a very even distribution of evaluations as to whether or not players felt like they had to employ different strategies

There is one more problem that was reported in the questionnaire: Players felt they had to repeat steps several times. This is because whenever an interaction is failed, if it made use of reward power, the player must then repeat all needed steps to reacquire the item.

Finally, there were some issues whose causes lie in the field of UI/UX:

(1) It was not communicated clearly enough to the player that using the same item on the same agent repeatedly would have no effect, so players tried that several times.
(2) Players can carry several of the same item, and did they do, carrying large quantities of an item even despite item producing no power when given in bulk.
(3) Players ignored the instruction to start by going to the vending machine, and tried instead to influence behaviours. This is likely due to that instruction being in the middle of a block of text, that playes easily miss.
(4) Sometimes players had trouble recognizing the consequences of an action, as they are only visible in the comparison of current current values with the previous current values for each base of power.
(5) When prompted to talk, if the agent has no lines to say, or a single line, there's no visual effect that communicates to the player that the reason the text box does not change is because there's no more text to show, and not that the button somehow didn't work. So we saw often players pressing the talk button repeatedly.
(6) When successfully influencing a behaviour, sometimes the player has performed several actions on the agent already, and it is not clear which ones were important. This leads to the player having to once again try to figure out how to influence that behaviour if they ever need it again.

8 FUTURE WORK

Most of our work on this project has been foundational. Both getting the interaction model working and devising a generation algorithm that is both flexible, and diverse. The solution we built allows for the easy focus on specific parts of the subject. One could now just work on the graph building algorithm and not touch any other part of the code, and everything should still work. As such, there are still lots of venues of inquiry that we could not explore in the limited time we had.

We can split possible future work on its scope:

(1) **Work on the model**, namely to address the issue of it being possible to brute-force a solution;
(2) **Work on the generator**, to address the lack of designer control, to address the "empty" agents that can be generated, and to increase performance;
(3) **Work on the game**, to address the issues of user experience that were raised during testing;
(4) **Work that would involve both the generator and game**, to expand on our solution, with new types of nodes, actions, behaviours and their integration in the game.

9 CONCLUSION

NPCs are a crucial element of many games, and providing a wider array of possible interaction with them can create more interesting gameplay scenarios. For this purpose, we have developed a model to guide NPC behaviour based on a theory of social power, and a generating algorithm that is capable of generating societies which present different problems for the player to solve making use of the affordances of that theory of power. The social power theory used is French and Raven's Bases of Power, described in chapter 2. Our work was split in three main components:

- The Interaction Model
- The Generating Algorithm
- The Test Game

As there were some concessions that had to be made to better fit the model into the context of a game, we implemented a simplified version of it that was reactive instead of based on expectations. In other words, while the original theory saw power as a function of the agent's expectations of being reward or coerced, our model sees power as something that is given directly in response to the players actions.

The generating algorithm was implemented in such a way as to easily allow modification of its parts, to permit flexibility in adapting it to different game specifications. It should function as a good foundation for further research into how to better apply this theory of social power to games.

The results of testing the overall project showed potential in the model as a basis for generative gameplay. Our system is capable of producing varied scenarios that create a social gameplay based on the theory of bases of power, with a diverse array of possible arrangements of different actions, behaviours and agents. Testing confirmed the adequacy and efficacy of both the underlying model and the generative algorithm. The generated scenarios are diverse, the generation algorithm flexible and controllable, and the mechanics present were understood by players, who had little difficulties completing the test scenarios. As for the game itself, which was not a major focus of the work in this dissertation, there were some questions regarding player experience that could be interesting to explore in further research.

REFERENCES

Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. 2011a. Interactive evolution for the procedural generation of tracks in a high-end racing game. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 395–402.

Luigi Cardamone, Georgios N Yannakakis, Julian Togelius, and Pier Luca Lanzi. 2011b. Evolving interesting maps for a first person shooter. In *European Conference on the Applications of Evolutionary Computation*. Springer, 63–72.

Agoston E Eiben, James E Smith, et al. 2003. *Introduction to evolutionary computing*. Vol. 53. Springer.

John R French, Bertram Raven, and Dorwin Cartwright. 1959. The bases of social power. *Classics of organization theory* 7 (1959), 311–320.

Matthew Guzdial and Mark Riedl. 2016. Game level generation from gameplay videos. In *Twelfth artificial intelligence and interactive digital entertainment conference*.

Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The unreasonable effectiveness of data. *IEEE intelligent systems* 24, 2 (2009), 8–12.

Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9, 1 (2013), 1–22.

Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. 2010. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. 1–4.

Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. 2019. Deep learning for video game playing. *IEEE Transactions on Games* 12, 1 (2019), 1–20.

Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. 2020. Pcgrl: Procedural content generation via reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 16. 95–101.

Joshua McCoy, Mike Treanor, Ben Samuel, Noah Wardrip-Fruin, and Michael Mateas. 2011. Comme il faut: A system for authoring playable social models. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Gonçalo Pereira, Rui Prada, and Pedro A Santos. 2016. Integrating social power into the decision-making of cognitive agents. *Artificial Intelligence* 241 (2016), 1–44.

Mark O Riedl and Robert Michael Young. 2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research* 39 (2010), 217–268.

Noor Shaker, Miguel Nicolau, Georgios N Yannakakis, Julian Togelius, and Michael O'neill. 2012. Evolving levels for super mario bros using grammatical evolution. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 304–311.

Noor Shaker, Julian Togelius, and Mark J Nelson. 2016. *Procedural content generation in games*. Springer.

Sam Snodgrass and Santiago Ontanón. 2016. Controllable Procedural Content Generation via Constrained Multi-Dimensional Markov Chain Sampling.. In *IJCAI*. 780–786.

Adam Summerville and Michael Mateas. 2015. Sampling hyrule: Sampling probabilistic machine learning for level generation. In *Proceedings of the Eleventh Artificial Intelligence and Interactive Digital Conference*.

Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270.

Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N Yannakakis. 2011a. What is procedural content generation? Mario on the borderline. In *Proceedings of the 2nd international workshop on procedural content generation in games*. 1–6.

Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2011b. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

Roland Van Der Linden, Ricardo Lopes, and Rafael Bidarra. 2013. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 1 (2013), 78–89.

Max Weber. 1978. *Economy and society: An outline of interpretive sociology*. Vol. 2. University of California press.