Universiteit Leiden

Master Computer Science

Dungeons & Firearms:
AI-Directing Action Intensity of Procedural Levels

Name:	Vincent Lucas Prins
Student ID:	s1935763
Date:	16/08/2023
Specialisation:	Artificial Intelligence
1st supervisor:	Mike Preuss
2nd supervisor:	Walter Kosters
3rd supervisor:	Matthias Müller-Brockhausen

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

ACKNOWLEDGEMENTS

First off, I would like to thank my first supervisor Mike for our brainstorm and feedback sessions surrounding the design of the game and research. Although I am not usually one for regular(ish) check-ins, it helped me keep the work focussed and concrete.

I would also like to thank Matthias and Walter for their valuable feedback while writing my thesis draft, helping to refine it. Additionally, both Mike and Matthias have helped me gain valuable academic experience during the last year of my Master's degree while we were working on the paper on procedural Minecraft settlements. That experience and the feedback received were great drivers to work on my thesis with the same vigour.

Jelmer and Laura have been really helpful with play-testing the game. Laura especially with her never-ending and meticulous feedback on precisely which shade of red the game should use.

Lastly, I am grateful to all participants who bravely ventured deeper into the dungeon.

Dungeons & Firearms: AI-Directing Action Intensity of Procedural Levels

Vincent L. Prins,
LIACS, Universiteit Leiden, The Netherlands
v.l.prins@umail.leidenuniv.nl

***Abstract*—Experience-Driven Procedural Content Generation is an extension to Procedural Content Generation, meant to personalise generative content to optimise user experience. In this paper, we present GONDVAAN, a procedural dungeon crawler and shooter game, where dungeons and firearms are procedurally generated. Our aim was to create an AI-director that could predict how action-intense procedural dungeons will be. We conducted two experiments with 11 and 16 participants respectively. The first one focussed on collecting play data for training an AI-director; 289 levels were played. The second one focussed on how certain game metrics are affected by levels of different AI-directed action intensities, as well as a gradual difficulty increase; 347 levels were played. We found that the AI-director could successfully present levels of the desired intensity, and that intensity and difficulty influenced certain game metrics.**

***Index Terms*—Procedural Content Generation, Player Experience Modelling, Dungeon Generation, Weapon Generation, Rogue-like**

I. INTRODUCTION

Gaming has reached a wide player base far beyond the classical stereotypical gamer. This change in audience comes with a great variety in their skills and preferences as well. Stark contrasts can be seen in the scene, from hard-as-nails Dark Souls to cheerful Animal Crossing. Game designers can already respond to this by providing adjustable difficulty settings or developing for a more niche audience, in the hopes that every player finds what suits them. This design process is, however, also time-consuming. Procedural Content Generation (PCG) may offer a partial solution in the form of large amounts of generative game content, but it is not instantly evident how this content relates to player-experienced aspects like difficulty, and particularly enjoyment: for instance, how gaps in a Mario level relate to how "fun" the player rates the experience [1].

Experience-Driven Procedural Content Generation (EDPCG), then, is a sub-field of Game AI that focuses on personalising the content of a generator for the sake of the user. Common uses are personalisation towards user preference or user skill level. The latter has been extensively researched in the platformer game genre [1]–[4], although racing games also has presence in research [5], [6].

This work presents GONDVAAN (Figure 1), meaning *battle flag*, derived from Old Dutch. It is a procedural dungeon crawler and shooter singleplayer game that can be placed in the rogue-like genre. One unusual feature of the game is that it presents the player with a choice between three procedurally generated dungeons and three procedurally generated firearms before the start of each level. We use this prototyped game as a test-bed for gameplay-based Player Experience Modelling (PEM). In particular, we focus on modelling *action intensity*, an ordinal value (low, medium or high) representative of the number of keyboard/mouse actions a player performs.

Fig. 1: Screengrab of GONDVAAN, a procedural dungeon crawler and shooter game.

Our research goals are to:

1) Setup a Machine Learning (ML) model trained on play data that predicts the action intensity of procedurally generated levels. This will be used as an AI-director.
2) Investigate the effects that presenting levels of different AI-directed action intensities to players has on various game metrics.
3) Investigate what effect increasing game difficulty has on various game metrics.
4) Investigate what kind of procedural weapons players prefer, and whether there is interplay between the chosen level and the chosen gun.

We will now proceed with an overview of related work in the domain of PCG and EDPCG (Section II). Thereafter, the design of the game GONDVAAN will be presented (Section III). We conducted two experiments with the game, linked to the research goals (Section IV). We then discuss the experimental results (Section V). Lastly, we will mention some weaknesses and potential future work (Section VI) and conclude this paper (Section VII).

II. Related Work

In this section we will provide an overview of related work in the domain of Experience-Driven Procedural Content Generation (EDPCG) and the procedural generation of weapons and dungeons.

A. EDPCG

Yannakakis and Togelius present a framework for EDPCG [7] including three types of Player Experience Modelling (PEM) depending on the data collection method used. These are *subjective* (e.g. a questionnaire), *objective* (e.g. heart-rate monitoring) and *gameplay-based* (e.g. game score) data collection. The latter is used in our work (Section IV-A).

In [8], Stammer et al. look at difficulty adaptation for the 2D platformer Spelunky. Specifically, game content in levels is adapted based on play style and player performance. In one experiment, players were assigned one of three play styles (explorer, speed runner or enemy killer) based on how an introductory level is completed. Next, the difficulty dynamically transitions between easy, medium and hard based on the performance of the player. This difficulty affects the probabilities of certain game content (e.g. items, enemies, traps) being generated. This is, however, also profile dependent. For example, hard mode for the enemy killer profile results in more enemies than hard mode for the speed runner profile.

Conversely, in our work, difficulty will merely affect the strength of existing enemies, not the quantity (Section IV-C). The AI-director will however be able to steer the number of enemies, as well as the quantities of other game content.

Bicho and Martinho [9] propose that player performance can be measured as a pairing of the type of challenge and the method used to overcome that challenge. In their platformer game, players are presented with six different challenges that each can be tackled with more than one method (e.g. an obstacle that can be jumped over, or slid under). Thus, when the success rate of a challenge-method pair is taken into account in PEM, challenge difficulty can be adjusted on a per method basis: a player great at jumping but poor at sliding will be presented hard jump challenges, but easy slide challenges. The PEM calculates this based on the success rate of the 10 most recent attempts (i.e. gameplay-based), with more recent attempts weighted more heavily.

B. Procedural Weapon Generation

Weapons are a good example of game content that can be procedurally generated. Some commercial games already include this, such as the Borderlands series [10] and Starbound [11].

In research, it also has prevalence. Brown [12] describes an evolutionary algorithm to generate new weapons, specifically one that can be integrated into a game story-wise as a form of interactive evolution. Similar to Borderlands, here too weapons can be made up of component parts that combine to form the final weapon. Crossover could then swap components between instances of the population, and mutation can alter a single component. Brown, however, notes that balancing such evolutionary weapons can be tricky, as ideally, players should not be able to create an all-powerful weapon early on.

Pace and Thompson [13] came across the same problem in their EvoTanks game environment. They too used an evolutionary algorithm to generate parameterised weapons, but also shields at the same time. Tanks fight against each other to test their fitness, and therefore the fitness of the weapon and shield used. At times, they found the evolved weapons to be "overpowering" shields, and at other times vice versa. Possible presented solutions include balancing the damage output per second, or working with a system of "points" allotment: set a maximum number of available points that can be spent on properties like damage and fire rate. To combine this with Brown's work: we suggest slowly increasing the maximum number of points that can be spend, as it allows for more powerful weapons to only come available later in the game. For the weapons generator in our work, however, we balanced firearms based on the damage output per second (Section III-C).

Galactic Arms Race is an online multiplayer game presented by Hastings et al. [14] in which the generative space of projectile weapons for spaceships is explored by players through an evolutionary algorithm. The motion of the projectiles is determined by a compositional pattern-producing network. These networks are evolved via a form of NeuroEvolution of Augmenting Topologies (NEAT) that takes into account the fitness of weapons as measured from player usage. Using a weapon increases its fitness, while decreasing the fitness of other weapons in the player's inventory. Players, however, only have three inventory slots to collect weapons. Thus, they must carefully choose which weapons to carry, and in doing so steer the evolution and exploration of the generative content.

C. Procedural Dungeon Generation

In their survey on procedural dungeon generation [15], Viana and dos Santos identify some of issues that current dungeon generators suffer from. These include a lack of difficulty adaptive generators and mixed-initiative approaches. Moreover, few works consider barriers in a level or try differentiate certain areas of the dungeon for a certain purpose (e.g. boss rooms, treasure rooms, etc.).

In our work, we present a method of adapting a dungeon generator to *action intensity* (Section IV-A). Additionally, some obstacles may be present in the levels, blocking access of parts of the level on occassion until they are attacked and destroyed by the player. This is, however, merely an artifact of the random placement of obstacles (Section III-B).

The approach of Sorenson et al. [16] focuses on *rhythm groups*: a chronological series of game content that alternates high and low levels of challenge. They first apply this to constructing 1D Super Mario platformer levels by means of a genetic algorithm. The same approach was then applied to 2D Zelda dungeons to showcase its generality. Here, rhythm groups are expressed as a sequence of dungeon rooms.

In our work, we will use an AI-director to insert rhythm groups in the gameplay on a level-to-level basis, but not within a level (Section IV-A).

III. Game Description

GONDVAAN is a procedural dungeon crawler and shooter game. At the beginning of each level, the player is presented a choice between three procedurally generated weapons and three procedurally generated levels and must choose one of each (see Figure 2). The player then enters the chosen dungeon and has to reach the exit (shown in-game as a ladder) to complete the level. In the meantime, they will be attacked by enemies. When killed, enemies drop *gems* used as a currency to open *upgrade chests*. These provide upgrades to player movement speed, gun reloading speed, and more. Akin to other rogue-like games, the player only has one life and the game can be played infinitely (if the player survives) due to its procedural levels.

Fig. 2: The selection screen shown before each level. Players must choose a gun and a dungeon to play.

The game design was mainly inspired from Enter the Gungeon [17]. Also of note is the game Rift Wizard [18], which happens to have a comparable level preview system. In this game, each level contains "rifts" that lead to other levels. Players are shown a preview of the level that waits in each rift before they actually enter and can thus make an informed decision on where to go next.

We will now discuss the various aspects of GONDVAAN.

A. Dungeons

Dungeons are defined as a 30×30 matrix of *tiles*. There are three tile types: empty, wall, and floor tiles. Dungeons are procedurally generated according to *construction steps*. A construction step defines how many rectangles of what tile type and size should be placed in the tile matrix. This placement overrides whatever was there before. The possible parameters of a construction step are given in Table I. Note how horizontal and vertical size are separate, allowing for rectangles.

TABLE I: Parameters of a *construction step*.

Parameter	**Value Range**
Number of rectangles	1–25
Tile type	Floor or Empty
Minimum horizontal size	1–5
Minimum vertical size	1–5
Maximum horizontal size	[minimum] + 0–5
Maximum vertical size	[minimum] + 0–5

To construct a dungeon, between two and five construction steps are randomly generated and executed. Executing a construction step means generating N rooms of the size and tile type as described by the parameters, and placing it at a random location within the matrix. For the horizontal and vertical size of the room, we generate a number between the minimum and the maximum size. This means a single construction step can already generate rectangles of different sizes.

After executing all construction steps, we try to place doors and walls to create rooms. This was, however, very much an afterthought in the design process, unlike for example the dungeon rooms in [16], so our method might seem unusual.

First, 100 attempts are made to choose random 2D points in the dungeon. An attempt succeeds if it selects a floor tile that is between 2 and 12 tiles away from a wall, and if it is sufficiently far away from another attempt's 2D point. The latter can be adjusted by the parameter *door distance* and is designed to space out doors and thus increase or decrease room sizes. Next, a door is placed at every selected point and adjacent floor tiles on either the x or y axis are recursively converted to wall tiles. In this way, rooms are created.

Lastly, we floodfill the floor (passing through doors) starting from the centre of the matrix and remove any floor tiles not covered in the floodfill (i.e. convert them to empty tiles). We then place walls around the floor tiles so the player cannot escape the dungeon. Lastly, we add *dungeon objects* (see Section III-B) and determine an entrance and exit location sufficiently spaced apart, so the player must explore the dungeon to exit it.

The algorithm described thus far can, however, generate invalid dungeons. For example, two construction steps with *tile type empty* will not create any floor. Alternatively, if a first construction step places floor tiles, the second step could replace all of it with empty tiles. To solve this, we apply a (very basic) form of constrained search-based PCG [19]. We use the number of floor tiles as a constraint: we require a minimum of 20% floor tiles and a maximum of 90%. A maximum is not a requirement for validity, but it avoids visually uninteresting levels (e.g. just a square of 30×30 floor tiles). We simply keep searching random parameters until the resulting dungeon meets our constraints.

Figure 3 showcases the variety that the dungeon generator is capable of creating. Each sub-figure uses different construction steps. In Figure 4, however, several dungeons constructed using the same construction steps are shown, illustrating how different random seeds following the same construction steps still lead to variety.

Fig. 3: Examples of dungeons with different construction steps and door distances.

Fig. 4: Examples of dungeons with the same construction steps and door distance.

B. Dungeon Objects

Once we have the general shape of the dungeon, we can populate it with game objects. Because the size of the dungeon can greatly differ, we work with object densities, as they are applicable regardless of dungeon size. Table II lists the dungeon objects and the possible densities in which they can occur (e.g. 1 chest per 75 tiles at its most dense). For each object, we choose a random value in its density range, calculate how many objects are needed to reach that density given the dungeon size, and then place those objects randomly.

The objects are as follows: *obstacles* can be used to hide behind but can also be destroyed by the player; *chests* can be opened at the expense of currency and provide upgrades to the player; *nests* spawn more enemies on a 60 second interval; *health and ammo packs* can be picked up to replenish the player; lastly, *enemies* are described in Section III-D. Figure 5 shows a finished dungeon, including its dungeon objects.

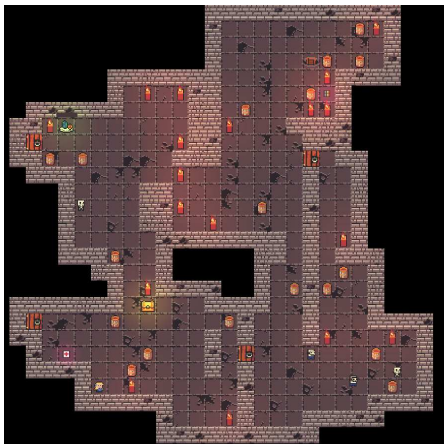

Fig. 5: A complete dungeon level with dungeons objects and enemies in place.

TABLE II: Density ranges of *dungeon objects*. To aid the reader, *instance ranges* are given, calculated to be the theoretical minimum and maximum number of instances of each object type.

Parameter	Density Range	Instance Range
Obstacles	15–35	5–54
Chests	75–300	0–10
Enemies	50–150	1–16
Buffs (Health & Ammo)	50–300	0–16
Nests	75–300	0–10

TABLE III: Parameters of a firearm.

Parameter	Value Range
Bullet speed	4–10
Fire delay (s)	0.2–0.7
Reload delay (s)	1–3
Bullets per shot	[1, 2, 3, 4]
Bullet spread (°)	2–7
Bullet colour	0.5–1 per RGB channel
Recoil	0.1–3
Burst gun	[True, False]
Burst shots	[2, 3]
Burst delay (s)	0.08–0.15
Star gun	[True, False]
Star separation speed	0.1–3
Rotating star	[True, False]
Rotation (°/s)	90–720

C. Firearms

Firearms in GONDVAAN are also parameterised. The parameters and their respective value ranges are shown in Table III. The generator samples uniformly randomly from this parameter space. All guns have a magazine that supports 10 shots. A single shot can have more than one bullet, as defined by the parameters. Bullets do not travel in a perfect line, but deviate with a certain spreading. There is a minimum delay between firing shots, and once empty the gun must be reloaded. If the "burst gun" parameter is true, the gun shoots out multiple bursts of shots with just one mouse click, but for the price of a single shot. If the "star gun" parameter is true, the N bullets in a shot are arranged in a circle $\frac{360}{N}^\circ$ apart, with a diameter that increases by "separation speed" per second, resembling a growing star shape. A boolean controls whether that star rotates. Lastly, bullets have a random colour for decorative purposes only. Figure 6 showcases some of the resulting guns.

Similar to Galactic Arms Race [14] and EvoTanks [13], we seek to balance the power of all generated weapons. In GONDVAAN, all weapons produce the same amount of damage per second (given a long enough time measurement). We determine how long it takes to empty the magazine of a loaded gun and how many bullets are shot in that process. From here, we calculate the number of bullets per second and subsequently adjust the damage of a single bullet. In the end, the bullets of a fast firing "machine gun" do less damage individually than those of a slow "shot gun" (i.e. firearm parameters that resemble those gun types, as we do

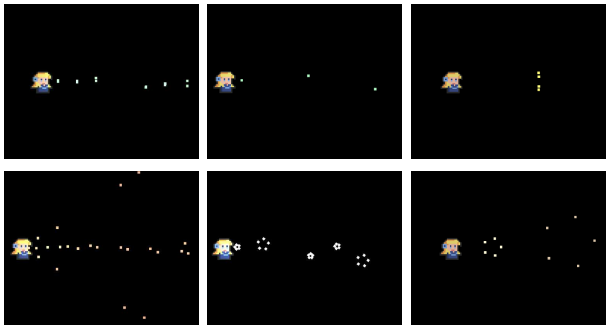

Fig. 6: Examples of procedurally generated firearms. The bottom row shows weapons with the "star gun" property.

not define these explicitly).

D. Enemies

For the enemies in GONDVAAN, we defined a generic AI controller that can be parameterised to display a range of behaviours. Table IV summarises these parameters. Enemies have a vision range and will remember where they last saw the player and health packs. When enemies are low on health, they will flee when approached and try to collect health packs, meaning they heal up and will no longer flee. If the player is spotted, enemies will approach the player up to a certain distance, and start shooting from a certain distance. Enemies are equipped with a random gun generated from the same firearms generator used by the player.

Similar to Pereira et al. [20], the enemy AI agents make use of *behaviour cycles*: each N seconds, an enemy chooses what to do until the next interval. When no player is in sight, enemies enter their idle cycle, otherwise they enter their attack cycle. In the idle cycle, enemies either stand still or move in a random direction until the next interval. However, if the enemy also has a preference for actively seeking out the player, it may move to the position where the enemy last saw them instead of a random direction. In the attack cycle, enemies choose whether to *strafe* (move left or right) for each interval. If no strafing occurs, enemies will simply maintain their approach distance.

Within this parameter space, we defined four enemy "species", somewhat similar to "player personas", but not focussed on adaptive PCG like in [21]. Instead, species define a value range for each parameter, from which we sample uniformly randomly. In this way, individual enemies of the same species can still somewhat differ in their specific behaviour, but overall, their personality traits will clearly represent the species. Further consolidating this, all enemies of the same species have the same in-game appearance. The four enemy species are:

The dive-bomber has little health and never flees. It moves very quickly and rarely stand still. It tries to get up-close to the player and only start shooting once close-by.

TABLE IV: Parameters of an enemy.

Parameter	Description
Health	The amount of health.
Flee Health	At what percentage of health to start fleeing and collecting health packs.
Reward	How many gems enemies drop upon death.
Accuracy	How accurate enemies aim their gun.
Vision	At what distance enemies can perceive players and objects.
Attack Distance	At what distance enemies start firing their guns.
Approach Distance	The closest distance an enemy will approach. When players try to get closer, the enemy backs off.
Move speed	The speed at which enemies move.
Idle Cycle	Duration in seconds of each idle cycle.
Stand Still Preference	At each new idle cycle, the probability an enemy will stand still that cycle.
Find Player Preference	At each new idle cycle, if not chosen to stand still, the probability an enemy will actively seek out the player's last know position.
Attack Cycle	Duration in seconds of each attack cycle.
Strafe Preference	At each new attack cycle, the probability an enemy will strafe. A random direction (left or right) is chosen for that cycle.
Strafe Strength	Interpolation value between the strafe direction vector and the vector pointing at the player. At 0, no strafing occurs; at 1, the enemy will only strafe and thus will not come any closer, instead just circling around the player.

The dodger stays at a medium distance from the player when attacking, and circles the player in an attempt to dodge incoming fire (using the strafing mechanic).

The shy inquisitor attacks from a long distance and may quickly flee even when it has received little damage. It is also rather inquisitive and may try to seek out the player once they are out of the vision range. It tends to stand still when idling.

The hitman has the most health of all species. Once the player has left its vision range, it will almost always try to actively seek out the player once more. It also somewhat tries to dodge in combat, but not as much as the dodger.

IV. EXPERIMENTAL SETUP

In this section, we will discuss our approach for each research goal and how they relate to the conducted experiments. Thereafter, the conducted experiments are detailed.

A. AI-Director Setup

The first research goal is to setup a Machine Learning (ML) classification model trained on play data that predicts ordinal categories of action intensity of procedurally generated dungeons. We will obtain play data from the first experiment.

The dungeon generator makes use of 13 parameters: those listed in Tables I & II, as well as the *number of construction*

steps and the *door distance*. However, as described in Section III-A, the translation from construction steps to dungeon shape is not always evident: future construction steps may erase any effect previous steps had. Thus, we gather spatial information about the dungeon *after* it has been constructed. This information is detailed in Table V and includes properties like the percentage of floor, wall and empty tiles; the largest square floor area of size $N \times N$; and more.

TABLE V: Spatial properties of the dungeon shape.

Property	Description
Percentage Floor	Percentage of floor tiles in the level
Percentage Wall	Percentage of wall tiles in the level
Percentage Empty	Percentage of empty tiles in the level
Horizontal Floor	Largest number of horizontally adjacent floor tiles.
Horizontal Wall	Largest number of horizontally adjacent wall tiles.
Vertical Floor	Largest number of vertically adjacent floor tiles.
Vertical Wall	Largest number of vertically adjacent wall tiles.
Square Floor	Largest square area of floor tiles (e.g. 5×5, so 5).

The action intensity of a level is determined as follows: first we count the numerical number of actions taken in a level. We take into consideration every action that requires a key press or mouse click. However, not all actions have the same subjective intensity: opening a chest is more exciting than walking to the left. Therefore, we employed action scalars (Table VI). Pressing a key to walk left adds 1 to the action total; opening a chest adds 200 "actions" to the total. The values were, however, chosen on a purely subjective basis of how meaningful we as designers might consider the action.

TABLE VI: Intensity of each game action.

Action	Intensity scalar
Walking	1
Shooting	5
Opening doors	20
Collecting ammo packs	20
Reloading	50
Collecting gems	50
Collecting health packs	75
Opening chests	200

Next, we convert it to an ordinal variable. Collected data will be split into tertiles (quantiles of $\frac{1}{3}$ of the data) based on the number of actions performed in each level, representing low, medium and high action intensity. The tertiles are calculated on a per player basis, in order to account for different play styles. For example, when playing the same dungeon with the same skill, someone who mashes keys will generate more actions than someone who carefully presses keys during play. If the tertiles would be calculated over all data, the key masher might appear to always play high intensity levels.

In summary, the ML classification model has 14 inputs: the spatial properties of a dungeon (Table V), the densities of dungeon objects (Table II) and the *door distance*. Its output is an ordinal value (low, medium, high) that represents the action intensity of that dungeon.

B. Influence of Action Intensity

The second research goal is to investigate the effects that presenting levels of different action intensities to players has on other game metrics. This will be investigated in the second experiment.

For the game metrics, we chose *number of enemies killed*, *time spent in level*, and *time spent engaged in combat*. For the latter, we note that being engaged in combat with one enemy is a different experience than being attacked by four at once. Hence, we sum up the duration that all enemies individually are engaged in combat. This could therefore be a larger value than *time spent in level*.

Before the second experiment, we trained a ML model on data from the first experiment in order to be able to "AI-direct" the intensity of a dungeon based on its spatial properties. We modified the game so that players would progress through different levels of action intensity in a cycle of three levels (i.e. low, medium, high, low, medium, high, etc.). For this we took inspiration from the *rhythm groups* of Sorenson et al. [16]. In order to achieve this, for each level we first generate a pool of 25 dungeons. Next, the ML model predicts the action intensities, and we choose the level predicted to be of the desired intensity with the highest prediction certainty.

C. Influence of Difficulty

The third research goal is to investigate the effect of difficulty on the aforementioned game metrics. To achieve this, in the second experiment, after every third level we increase the difficulty: a scalar applied globally to enemy parameters (e.g. an increase in health or movement speed) and the currency cost to purchase upgrades from chests. In this way, within a cycle of three levels, only the intensity changes, but throughout the game the difficulty goes up as well. Difficulty starts at 1 and increases in steps of $\frac{1}{9}$ per cycle: an arbitrarily chosen value.

D. Weapon Preference

The fourth research goal is to investigate what kind of weapons players prefer, and whether there is interplay between the chosen level and the chosen gun. For this, we will combine the play data on firearms of the first and the second experiment. This consists of the parameters of all generated weapons, whether they were chosen by the player or not, and what corresponding dungeon was chosen at that time.

E. Experiments

In summary, in the first experiment, 11 participants played 289 procedurally generated dungeons from random points in the generative space. In the second experiment, 16 participants played 347 levels that were AI-directed to be of certain action intensity. Additionally, the game difficulty slowly increased in

the second experiment (but it would reset on player death). In total over two experiments, this means that 1908 instances of both dungeons and firearms were presented to players, three at a time, and 636 of these were chosen to play with.

Participants were provided a download link of the game and were asked to complete the in-game tutorial and to play as many levels as they liked. The game was modified to keep track of play data and save it. Play data was then sent to the researchers. Only completed levels were recorded. If the player died, a new session could be started as often as one liked.

V. Results

The results will be divided once more per research goal.

A. AI-Director

In the first experiment, 11 participants played 289 levels in total. With the number of actions split in tertiles per player, this resulted in 97 levels of low, 92 of medium and 100 of high intensity.

A decision tree from the SharpLearning C# library [22] was trained on the spatial dungeon properties and respective action intensity. The trained model had a test error of 44.29%. Its confusion matrix can be seen in Figure 7, showing all 289 levels. In a small test of 9 pools of 25 dungeons (3 per intensity category), its maximum prediction certainty averaged at 92.4%. Moreover, the generated levels of different intensities look visually distinct as shown in Figure 9. Thus, we determined that the model could be used for the second experiment. The relative feature importance of the trained model is shown in Table VII.

In the second experiment, 16 participants played 347 levels in total. Of those, the AI-director meant for there to be 131 low intensity levels, 118 medium, and 98 high intensity. To determine whether that intensity was actually met, we once more split the data in tertiles based on the number of actions, like in the first experiment. In order to rule out influence of difficulty on the action intensity, tertiles are calculated separately for each difficulty value. The resulting confusion matrix can be seen in Figure 8. Here, we see that 218 levels (62.8%) were at the correct intensity, 111 (32.0%) were off by one, and 18 (5.2%) were off by two.

TABLE VII: Relative feature importance of the trained decision tree. Here, the feature importance is the relative number of samples that reach a decision node of said feature.

Feature	Importance (%)
Percentage Floor	100.0
Enemy Density	63.7
Buff Density	37.4
Nest Density	24.5
Chest Density	22.9
Percentage Wall	21.5
Obstacle Density	16.6
Door Distance	9.4
Square Floor	9.1
Horizontal Wall	6.2
Vertical Wall	5.2
Percentage Empty	2.9

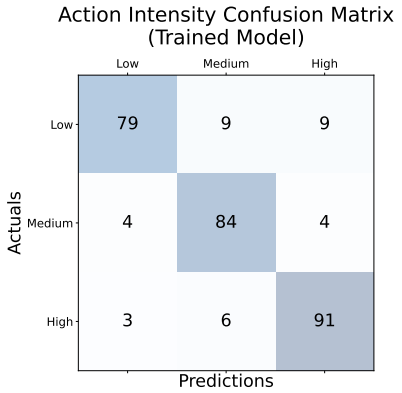

Fig. 7: Confusion matrix of the trained model on all 289 levels (both train and test sets) of the first experiment. We include all levels here for sake of comparison with Figure 8.

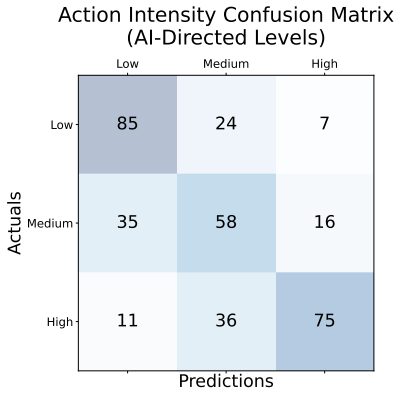

Fig. 8: Confusion matrix between the predicted action intensity of AI-directed levels in the second experiment and their actual intensity as determined from play data.

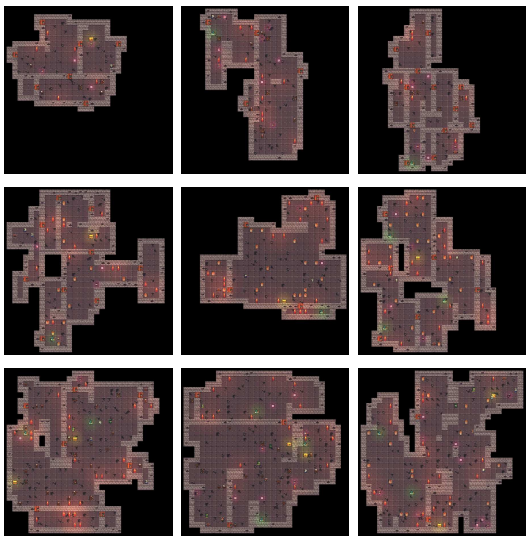

Fig. 9: Examples of dungeons at different predicted action intensities. Top row shows low, middle row shows medium, and bottom row shows high intensity.

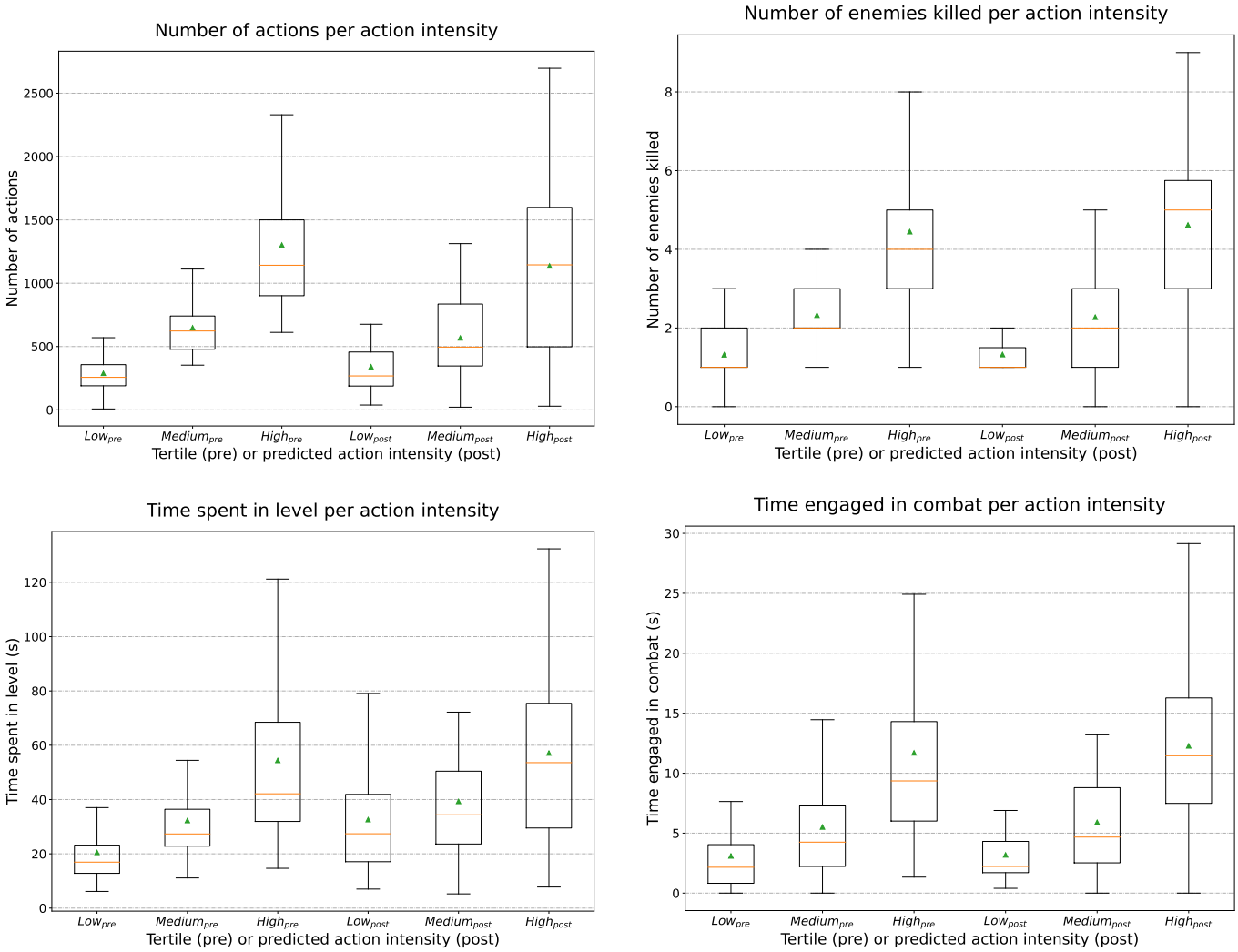

Fig. 10: Various player statistics plotted against the intensity of that level. Boxplots drawn with interquartile range 1.5 and outliers discarded. The triangle shows the mean, the line shows the median.

B. *Influence of Action Intensity*

For the second research goal, we look at the influence of action intensity on game metrics. The correlations between action intensity and game metrics are shown in Table VIII. The results are split up in pre and post AI-director. The results of the first experiment (pre-director) are split in ordinal categories based on the low, medium and high tertiles of the number of actions. For the second experiment (post), the results are split based on the AI-director's prediction of the level intensity, regardless of whether it was correct or not.

In Figure 10, the game metrics are shown as boxplots per action intensity category. Our first observation is that the AI-directed results (post) resemble the results from the first experiment. We also observe that when players are presented a level of higher predicted intensity by the AI-director, they on average spend more time in the level, in combat, and kill more enemies.

TABLE VIII: Pearson correlation between action intensity and game metrics.

	Action Intensity (pre)	**Action Intensity (post)**
Number of actions	0.72	0.53
Time in level	0.52	0.42
Time in combat	0.48	0.37
Enemies killed	0.61	0.63

C. *Influence of Difficulty on Metrics*

For the third research goal, we look at the influence of difficulty on game metrics. Game difficulty determines how strong enemies (e.g. their maximum health) are and how expensive upgrades are to purchase. In the first experiment, the difficulty of the game was fixed at 1. In the second experiment, it increased every 3 levels. Table IX shows how many levels were played per difficulty in the second experiment. In total,

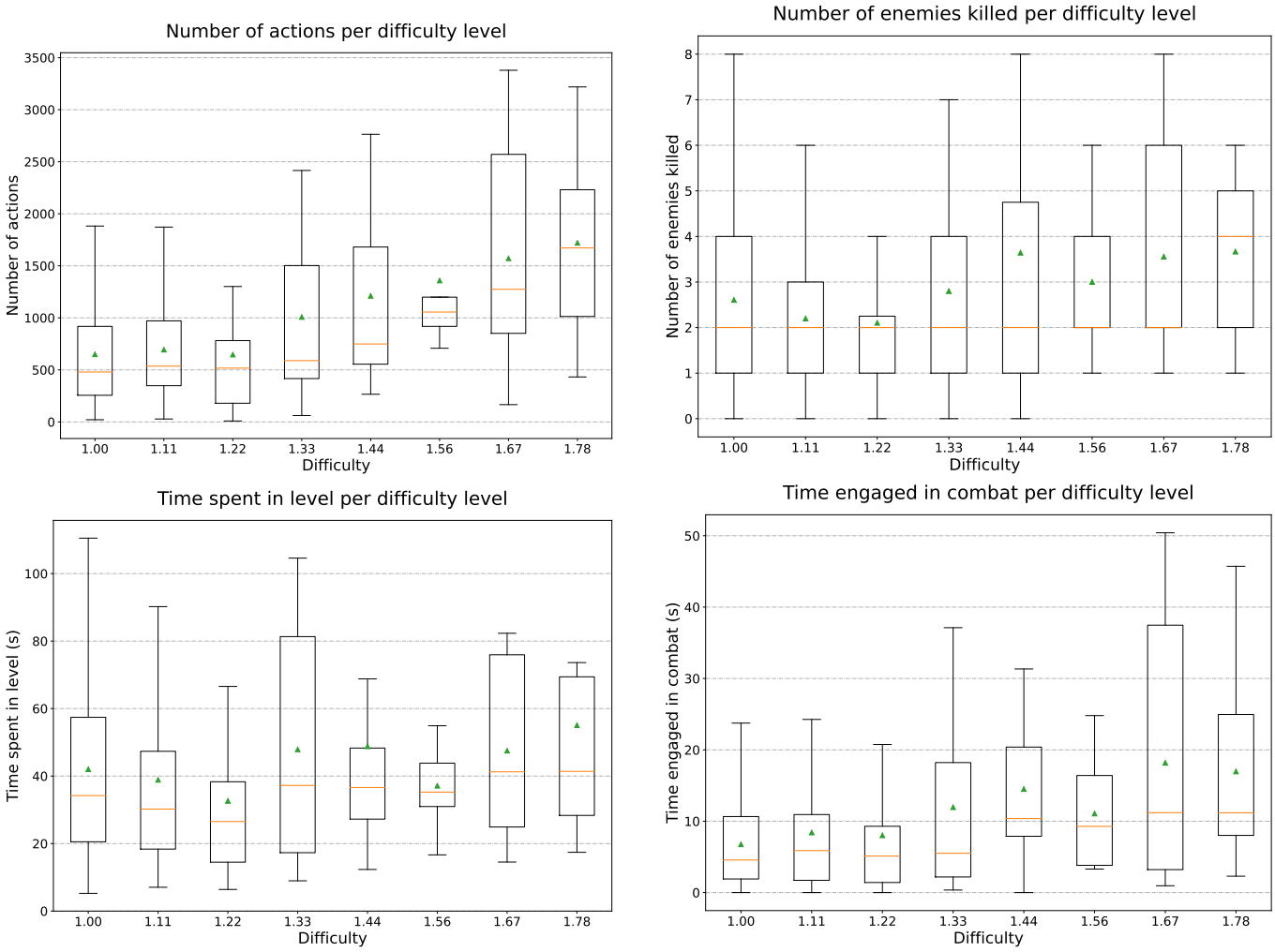

Fig. 11: Various player statistics plotted against the difficulty of that level. Boxplots drawn with interquartile range 1.5 and outliers discarded. The triangle shows the mean, the line shows the median.

117 levels were played at difficulty 1, and fewer for each difficulty above. Furthermore, 30 levels were played above difficulty $1\frac{7}{9}$, up to $2\frac{4}{9}$. In Table X, the correlations between difficulty and game metrics are shown.

TABLE IX: Number of levels played per difficulty.

Difficulty	1	$1\frac{1}{9}$	$1\frac{2}{9}$	$1\frac{3}{9}$	$1\frac{4}{9}$	$1\frac{5}{9}$	$1\frac{6}{9}$	$1\frac{7}{9}$	$> 1\frac{7}{9}$
Levels Played	117	81	48	30	14	9	9	9	< 9 per difficulty

TABLE X: Pearson correlations between difficulty and game metrics.

	Difficulty
Number of actions	0.44
Time in level	0.11
Time in combat	0.38
Enemies killed	0.16

Figure 11 shows the relation between difficulty and game metrics in detail as boxplots. Here, we have left out difficulty levels with fewer than 9 samples. Firstly, we observe a rise in the number of actions when the difficulty increases. We presume that more difficult enemies warrant more player actions to be defeated. Secondly, there is a small increase in the number of enemies killed as difficulty increases. Interestingly, note that difficulty has no influence on the number of enemies spawned, merely the power of those enemies. A possible explanation could be that players who kill more enemies are more likely to reach higher levels of difficulty. Thirdly, no strong trend is seen in Figure 11 in the time spent per level when difficulty increases, matching the weak correlation. Lastly, in more difficult levels, players seem to spend more time engaged in combat. This adds to the idea that player have to perform more actions (and spend more time) to defeat enemies at higher difficulty levels. Overall, these results indicate that the game difficulty setting did have influence on game metrics.

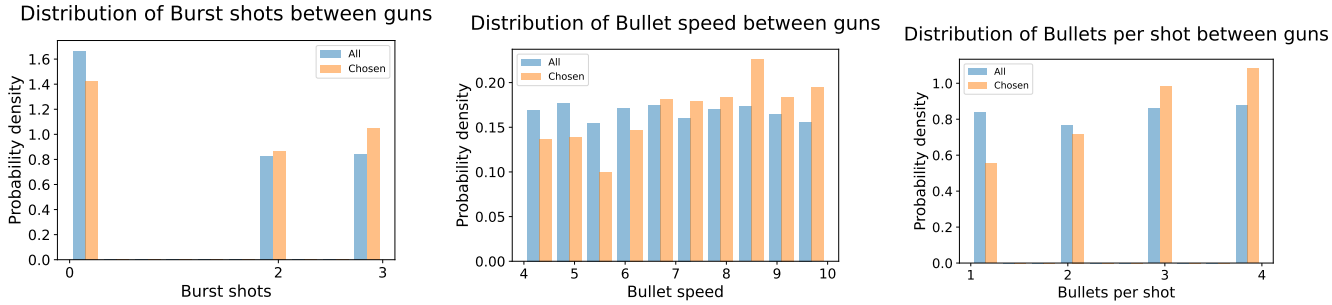

Fig. 12: Comparison between the probability densities of various firearms parameters in all guns and in player chosen guns.

D. Weapon Preference

The fourth research goal is to investigate what kind of weapons players prefer and whether there is interplay between the chosen level and the chosen gun. For this, we combined the data on firearms of both experiments. A total of 1908 firearms were generated, three at a time, and shown to players. Players chose 636 weapons to play with. The correlations between whether a weapon was chosen and its parameters are shown in Table XI.

TABLE XI: Pearson correlations between firearms parameters and whether the gun was chosen or not. Only $|r| > 0.1$ parameters are shown.

	Gun chosen by player
Bullet speed	0.13
Bullets per shot	0.15
Burst size	0.11

Overall, the correlations are weak, with only three parameters having $|r| > 0.1$. Figure 12 shows these in more detail. Here, the distribution of all weapons is shown against the distribution of the chosen weapons for these three parameters. Assuming these weak correlations indicate a trend, as seen from the skewed distributions, we could hesitantly say that players preferred weapons that featured the "burst" option (the more burst shots, the better), as well as guns that shoot many bullets, which travel at high speed.

Lastly, we investigated whether there was any interplay between the chosen level and the chosen gun. For each firearms parameter, we looked at which dungeon property it correlated with most strongly. The properties taken into account were: the spatial properties of the dungeon (Table V), the densities of dungeon objects (Table II) and the *door distance*. In the end, only one correlation $|r| > 0.1$ was found: "fire delay" and "longest horizontal wall" correlated at -0.11. Figure 13 shows this in further detail. Still, we found there to be little evidence to suggest interplay between dungeon and gun choice.

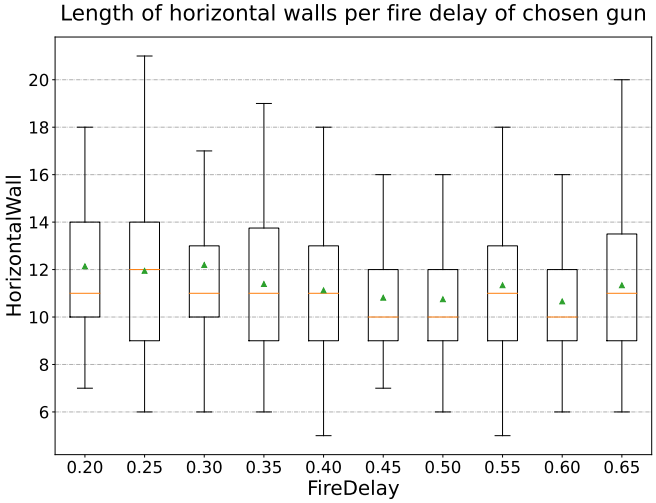

Fig. 13: The lengths of the largest horizontal walls plotted against the fire delay of the chosen gun in that level. Boxplots drawn with interquartile range 1.5 and outliers discarded. The triangle shows the mean, the line shows the median.

VI. Discussion & Future Work

One consideration of this work was to design a game that participants would enjoy playing, generating play data in the process. For the sake of gameplay, users could purchase upgrades like increased movement speed or quicker gun reload speed. However, the longer players play, the more upgrades they attain. We did not investigate how this could affect game metrics, especially in extended sessions. However, as seen in Table IX, 70.9% of all data is from level 9 or lower (where difficulty is $\leq 1\frac{2}{9}$) indicating a short session.

Furthermore, to investigate the effect of game difficulty, we increased the difficulty by $\frac{1}{9}$ after every third level. This may have been too little to be noticeable, requiring extended play sessions to generate enough data. Still, we were able to gather some results as-is.

A. Dungeons & Firearms

The generator for firearms in GONDVAAN, having 13 different parameters, could have been extended upon. Hastings et al. [14] note that ideal parameterisations should be able to surprise their designers. We did find some firearms with the "star gun" parameters to be quite interesting visually and mechanically (rotating bullets can curve around walls), but in the end rotating bullets were still a designer-specified feature. Still, there is an opportunity to vary bullet movement even more

without resorting to compositional pattern-producing network as in [14]. For instance, a new parameter could influence bullet movement with (co)sine or other cyclic functions. Lastly, we found it hard to design parameters that keep the power of guns in the generative space balanced and that do not defy what it means to be a firearm; for game designers, this may be a stylistic limitation.

In this work, we did look at player preference in weapons but not at preference in dungeons, mainly because we already sought to AI-direct level intensity. For player enjoyment, however, we do not know whether we should focus on directing the flow of the game, or present players their preferred levels.

Although future work in GONDVAAN could apply Player Experience Modelling (PEM) to firearms as well, we are sceptical as the evidence towards preference in certain parameter settings was not particularly strong. Expanding the firearms parameter space, as mentioned, could perhaps still warrant an interest in PEM with regards to firearms, though.

B. Enemies

In GONDVAAN, there are four enemy types. They all use the same generic enemy AI controller, but are parameterised differently. With these different parameterisations, we sought to create designer-specified "species".

Previous work, however, has looked at automating the procedural generation of parameterised enemies. For instance, Pereira et al. [20] use an evolutionary algorithm to generate different enemies according to their difficulty. The genotype represents enemy parameters like their health, movement speed and enemy weapons. The fitness, then, is some function of the parameters meant to represent the enemy's difficulty.

Similarly, we could search the parameter space of GONDVAAN enemies, but for species. A basic approach could restrict itself to a low and a high value for each parameter and present each combination of parameters as a species. Some species might be able to better counter the player's play style than others. This presents an opportunity for PEM.

Additionally, for now we merely added multiple enemy species in an attempt to make the game more engaging for participants. The enemies presented in each level were chosen at random. However, we did not investigate whether enemy species contribute to the game metrics differently.

VII. CONCLUSION

In this paper, we presented GONDVAAN: a procedural dungeon crawler and shooter game. In GONDVAAN, players are shown previews of three dungeons and three guns at the start of every new level, from which they can make their own choice. We used this game as a test-bed for Player Experience Modelling (PEM) and AI-directing. We focussed on four research goals: whether a Machine Learning (ML) model could predict the "action intensity" of levels based on its properties; how action intensity influences certain game metrics; how game difficulty influences those same metrics; and lastly, what firearms players prefer and how that choice relates to level choice.

We conducted two experiments, with 11 participants and 16 participants. In these experiments, 289 and 347 levels were played respectively. For each level, players also chose a firearm to play with. Since players were presented three dungeons and firearms to choose from at a time, 1908 were presented of both in total.

We found the ML model to successfully differentiate between three ordinal action intensities, making it suitable for AI-directing the game to have dungeons of different intensity levels.

Furthermore, we found action intensity to correlate with the number of enemies killed, time spent in a level, and time spent engaged in combat. The same can be said for game difficulty, however, here we are more cautious, as participants did not play as many levels over a wide range of difficulties as we had hoped for. Thus, results are indicative at best.

Early results also indicate that there might be a preference for certain types of guns, namely those with many bullets shooting in bursts with high bullet velocity; this, however, is not strongly backed-up and might merely indicate a trend. Moreover, we found no evidence pointing towards interplay between the choice of gun and the choice of level.

In conclusion, we deem it possible to AI-direct games with a model solely trained to predict the number of game actions: a highly generic metric that looks to be applicable to a wide range of game genres. We thus hope to see further research in other game genres that uses this method.

REFERENCES

[1] C. Pedersen, J. Togelius, and G. N. Yannakakis, "Modeling player experience for content creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 54–67, 2010.

[2] K. Zhang, J. Bai, and J. Liu, "Generating game levels of diverse behaviour engagement," in *2022 IEEE Conference on Games (CoG)*, 2022, pp. 167–174.

[3] N. Shaker, G. Yannakakis, and J. Togelius, "Towards automatic personalized content generation for platform games," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 6, no. 1, pp. 63–68, 2010.

[4] R. Parekh, "Staying in the flow using procedural content generation and dynamic difficulty adjustment," Ph.D. dissertation, Worcester Polytechnic Institute, 2017.

[5] A. Rietveld, S. Bakkes, and D. Roijers, "Circuit-adaptive challenge balancing in racing games," in *2014 IEEE Games Media Entertainment*, 2014, pp. 1–8.

[6] J. Togelius, R. De Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *2007 IEEE Symposium on Computational Intelligence and Games*, 2007, pp. 252–259.

[7] G. Yannakakis and J. Togelius, "Experience-driven procedural content generation," *IEEE Transactions on Affective Computing*, vol. 2, pp. 147–161, 2011.

[8] D. Stammer, T. Günther, and M. Preuss, "Player-adaptive spelunky level generation," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015, pp. 130–137.

[9] F. Bicho and C. Martinho, "Multi-dimensional player skill progression modelling for procedural content generation," in *Proceedings of the 13th International Conference on the Foundations of Digital Games*, ser. FDG '18. Association for Computing Machinery, 2018.

[10] Gearbox Software, "Borderlands," 2K, 2009.

[11] Chucklefish, "Starbound," Chucklefish, 2016.

[12] J. A. Brown, "Evolved weapons for rpg drop systems," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 2013, pp. 1–2.

[13] A. Pace and T. Thompson, "Procedural content generation and evolution within the evotanks domain." in *Proceedings of the 8th International Conference on the Foundations of Digital Games*, 2013, pp. 439–440.
[14] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Automatic content generation in the galactic arms race video game," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 4, pp. 245–263, 2009.
[15] B. M. F. Viana and S. R. dos Santos, "A survey of procedural dungeon generation," in *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2019, pp. 29–38.
[16] N. Sorenson, P. Pasquier, and S. DiPaola, "A generic approach to challenge modeling for the procedural creation of video game levels," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 229–244, 2011.
[17] Dodge Roll, "Enter the Gungeon," Devolver Digital, 2016.
[18] D. White, "Rift Wizard," http://www.dylanwhitegames.com/riftwizard/, 2021.
[19] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 3, pp. 172–186, 2011.
[20] L. T. Pereira, B. M. F. Viana, and C. F. M. Toledo, "Procedural enemy generation through parallel evolutionary algorithm," in *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2021, pp. 126–135.
[21] P. M. Fernandes, J. Jørgensen, and N. N. T. G. Poldervaart, "Adapting procedural content generation to player personas through evolution," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021, pp. 1–9.
[22] M. Dabros. (2020) SharpLearning 0.31.8. [Online]. Available: https://github.com/mdabros/SharpLearning

REFLECTIONS

It was with much enjoyment, but not without hick-ups, that I worked on this thesis.

Early on, when I was programming firearms and the early beginnings of dungeons, I was worried that my research would be too vague, and I had merely set myself the goal to apply Experience-Driven Procedural Content Generation (EDPCG) to firearms. I was also constantly thinking of how I could differentiate my work from that of Hastings et al., which together with Enter The Gungeon sparked my initial research proposal. Thus, I am glad that I started out reading a wide range of EDPCG research, as when I found that Player Experience Modelling firearms and interplay with dungeon choice might not lead the desired results, I was able to connect the dots and add a new avenue: modelling action intensity of dungeons instead. I am quite satisfied with how this ended up being, if may so say myself, rather successful: I got much personal enjoyment from playing in a rhythmic manner and it felt modelled correctly.

When designing a game for a thesis, one might be wary of focussing too much on game design, instead of the research at hand. I certainly was; so much so that I did not necessarily want to spend too much time on "refinements", focussing on practicalities instead. With some much needed pointers and encouragement from others these refinements in fact turned out to be valuable topics of research themselves: I refined the enemies, making their AI more intelligent and sophisticated, adding a parameter space and enemy species. Unknowingly, I now created an interesting new research paradigm in my game, which I only realised once helpfully pointed out to me that it was indeed so.

All in all, I will certainly fondly look back on Gondvaan.