Ministry of Education and Science of Ukraine
National University of "Kyiv-Mohyla Academy"

Informatics department of the Faculty of Informatics

Procedural content generation: resolving of customer satisfaction problem

Text part of course work

in speciality “Computer Science and Information Technologies” 112

Coursework supervisor
Assistant
Shabinsky A.S.

_____
(signature)
“____” _____ 2020 yr.

Made by student
Vasylenko K.A.
“____” _____ 2020 yr.

Kyiv 2020

Ministry of Education and Science of Ukraine
National University of "Kyiv-Mohyla Academy"

Informatics Department of the Faculty of Informatics

APPROVED

Head of the Informatics Department
associate professor, candidate of physical and mathematical sciences
_____ S.S. Gorokhovskyi
(signature)
"____" _____ 2020 yr.

INDIVIDUAL TASK
for course work

For the student of the Faculty of Informatics of 4 course of studying
THEME Procedural content generation: resolving of customer satisfaction problem

Output data:
Text part content of coursework:
An individual task
Calendar plan
Annotation
An introduction
Explanation of game balancing and customer retention and acquisition
Definition of the game and its problem
Discussion about Reinforcement learning and Unity as a learning platform
Learning process explanation
Conclusion
References
Applications

Issue date "____" _____ 2019 yr. Supervisor _____
(signature)
Task received _____
(signature)

Theme: Procedural content generation: resolving of customer satisfaction problem

Calendar plan of coursework execution:

№	Stage name	Deadline	Note
1.	Getting of coursework topic	01.09.2019	
2.	Searching of appropriate literature	10.09.2019	
3.	Research in-game balancing approach area	10.12.2019	
4.	Exploration of the puzzles subject area	25.12.2019	
5.	Exploring of technologies and frameworks	01.01.2020	
6.	Research in Unity Development	7.01.2020	
7.	Game specification definition	20.01.2020	
8.	Documentation formation	01.02.2020	
9.	Application's core implementation	10.03.2020	
10.	Writing game definition part	12.03.2020	
11.	Coursework analysis with the supervisor	13.03.2020	
12.	Writing learning part	20.03.2020	
13.	Coursework analysis with the supervisor	21.03.2020	
14.	Writing coursework summary	27.03.2020	

15.	Coursework analysis with the supervisor	28.03.2020	
16.	Coursework changing according to the supervisor's remarks	03.04.2020	
17.	Creation of the presentation	10.04.2020	
18.	Presentation of the coursework	19.04.2020	

Student Vasylenko K.A.

Supervisor Shabinsky A.S.

"____" _____

Contents

Abstract

Procedural content generation (PCG) is used in a variety of products for different purposes. Most techniques are hidden due to commercial reasons, which forces engineers to invent PCG algorithms from the beginning for every concrete problem. Apart from concrete purpose, current work has an aim to create a public solution for a single problem, which is common in thousands of games.

Firstly, this paper will define the specific benefits of PCG for a puzzle game. Problems that will be considered are customers retention, customers acquisition, session duration control.

The second aim is to implement a suitable algorithm for a game that will solve problems above.

Keywords - Procedural content generation, dynamic procedural content generation, customer retention, customers acquisition, session duration control, reinforcement learning, unity platform.

Introduction

Procedural content generation (PCG) is a technology used for the generation of media[1]. Which could be both music, poetry, drawing for some purpose and textures, sounds, levels, maps and so on for games. There are two types of PCG: Static and Dynamic, each type has a specific set of purposes. Firstly, dynamic generation is a generation of content during the execution of the game, commonly it depends on customers' input or reaction. On the other hand, static generation is any content produced before runtime such as maps, levels and so on.

There are two types of content to generate: *technical*, which directly influences game mechanics and *visual* that serves as a part of a user interface.

In addition, PCG for games has a classification [1]:

1. Game Bits - fundamental units of game content like pixels in pictures. This type includes, but not limited to textures, sound, sound effects.
2. Game Space - environment for games such as maps, terrain.
3. Game Systems - simulate complex environments, entity interactions.
4. Game Scenarios - organizations of levels, stories and puzzles.
5. Game Design - set of rules and mechanics of the game.
6. Derived Content - extra items to a game world like leaderboards or news.

Current work will discover dynamic procedural content generation of game design elements. The game has a block puzzle genre. Currently, the block puzzle means a game where a customer is given an N x N grid and M different types of puzzles. There are some amount of generated puzzles to put in a grid out of M

unique types. Aim of the game is to fill the grid using dynamically generated puzzles. When the user collects a row or column of N puzzles they will disappear. Game is endlessly remaining until there is no way to put the next puzzle in place.

Apart from a visual component, the core of the game lies in the generation of puzzle elements, which ought to imply on factors including:

1. Game balance
2. Customers retention
3. Customers acquisition
4. Session duration

Of course, a straightforward way to tackle the first problem may be a creation of difficulty levels and providing an ability to choose for customers, which is quite inefficient with a variety of players, who have extremely diverse skills together with different learning approaches. Thus, a solution is dynamic game balancing, which will be able to adapt as firmly as possible for every specific case.

Moreover, the procedural way of balance control provides an opportunity not only to control difficulty but also a session duration, which will be quite useful in satisfaction problem-solving.

The next section will explore the question of game balancing. Afterwards, customer retention together with the acquisition problems will be discussed as well. The final sections of work will dive into a case study and define a way of solving problems above together with an algorithm implementation.

1. Game Balancing

A game player's satisfaction is influenced by different variables, like the graphical interface, the background story, the input devices, and, in particular, game balancing. Game balancing aims at providing a good level of challenge for the user and is recognized by the game development community as a key characteristic for a successful game (Falstein 2004) [2].

1.1. Dynamic Game Balancing

Dynamic game balancing is a procedure that requires at least three fulfilled basic conditions. [2] First, the game ought to determine the customers' entry skills and experience, which obviously vary from a beginner to an advanced user. For instance, a novice in puzzles will not be able to understand most of the features at first and have to receive more long and easy game sessions, while an expert could become easily bored by a big deal of uncomplicated tasks that are only time-consuming. Second, players' performance in learning has to be tracked for providing an instant adaptation to acquired customers' skills. Nowadays it is not a secret that every person is diverse from others in his learning abilities and fast reaction to progress is a necessary ability. Third, adaptation has to be fluent and believable, so the player is not expected to feel that a machine is just adjusting parameters behind his eyes with an aim of improving his experience that will be quite disappointing rather than successful.

Many different approaches [2] to achieve a game balance have a mandatory function called "challenge function". Which measures a current difficulty that a user experiences by exploring game environmental parameters.

Hunicke and Chapman’s [3] approach for adjusting the difficulty of a gaming environment. For instance, for tough game experience players receive more bonuses like weapons, extra life points and fewer opponents, which is quite useful for games that have an ability to provide such bonuses. On the other hand, a board game has no ability to use such an approach.

The next approach is a modifying of Non-Player Characters' (NPC) behaviour which is usually controlled automatically and serves as intelligent agents. These agents typically use behavioural rules that are predefined in the development stage and switch between them based on domain-specific knowledge. In addition, there is a modification of this approach called dynamic scripting [4]. Dynamic scripting is a set of rules for an NPC, where each rule has a probability (weight) for being used. Weights are updated dynamically during the game process, so an agent is able to easily adjust to a players' skills. However, the harder a game, the more rules needed to keep into account, implementation harder. In addition, an agent's progress is limited by the best rule, so highly skilled players could easily become bored.

Finally, the last approach is teaching the agents with machine learning. With some predefined models at the beginning, the agent is improving through the game together with the player. Despite the convenience of this approach, it has limitations as well as other ones. For instance, as in the previous case, the agent's progress is restricted, so it cannot tackle uncommon behaviour or highly skilled users. Moreover, the agent is not able to degrade its progress in case of player's

skills degradation. As a result, it is a good strategy for providing a beginner on his way to become a pro rather than entertain sophisticated customers.

1.2. Progress-Sensitive Game Balancing

Another approach is a challenge-sensitive game balancing [2], which is usually implemented using reinforcement learning. Instead of predefining a specific behaviour to an agent or limiting his abilities to degrade, there is a way to teach an agent in advance. The main idea is to provide an ability to instantly adjust to a user's progress and play at the appropriate level at any time. Which requires solving of two tasks: competence - achieve as well as possible and performance - act as well as necessary. So, agents will be able to adapt to players' level including both capturing skills degradation cases and tackling with high performers.

Firstly, competence ought to be solved using reinforcement learning by playing with itself which is called self-learning. Also, it could be done by training with other intelligent agents. Second, performance is solved by providing a correct policy for choosing rules for game balance creation.

Current work is planned to use the progress-sensitive game balance in a way of creating an agent that first ought to learn how to solve the puzzle problem from any beginning stage. In addition, the agent has to detect whether it is possible to continue the game from a specific stage. Because of the reason that there is no end to the puzzle game, an agent needs no more than the ability to go ahead for N steps. As a result, an agent is available at any time for customers supporting during the game. But, there are some questions including: "What is support?" and "What if the customer does not have support?". Which will be discussed in detail in the implementation section.

2. Customer retention and acquisition

2.1. Explanation

Retention on day N is a percentage of customers who used a service again on the day N. Usually, in an industrial case it is better to measure the retention on day 1, day 3, day 7 or day 30. For instance, day 30 is 30 days after the retention graph's beginning [5].

For instance, if an application has 10 thousand customers on day 1, which is 100% then if it drops to 8000, day's 1 retention will be 80%. In case active users' amount dropped down to 4000 on day seven, so 4000 users out of 10000 came back on this day. Thus, retention means the percentage of users who used the application again on day N.

For a real example, retention of a game Bell Bird by Bellkross [6] will be demonstrated below.

Figure 2.1.1 - Retention table

On figure 1, on January 26, there were 3 138 total users that entered the app. 38.4% of which had come back after one week, that is 1205 unique users. So, the picture shows 61.6% users lose. In addition, users lose is increasing by each week and values of churn (opposite value to retention [7]) are 0%, 61.6%, 79.9%, 86.3%, 89.4% and 92% accordingly.

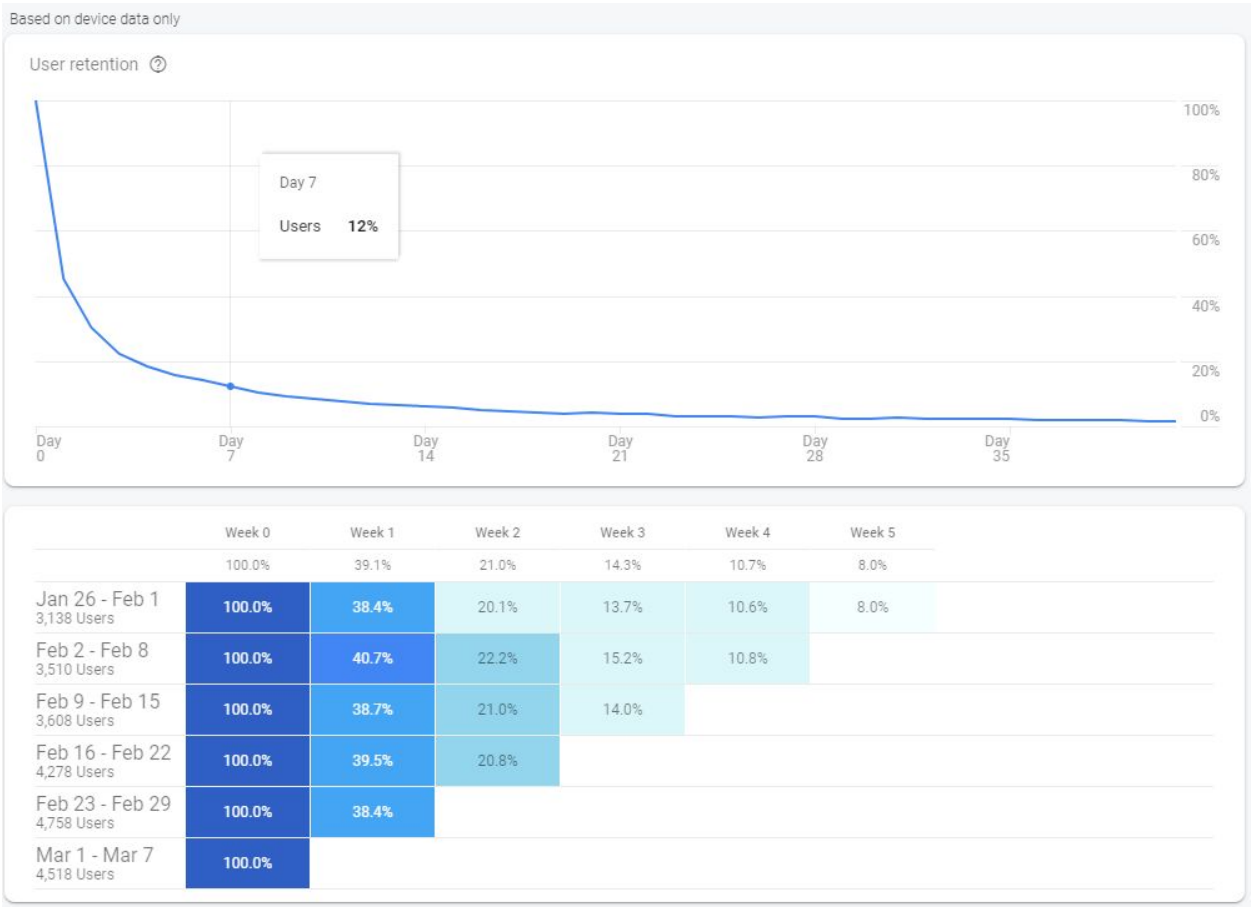

Figure 2.1.2 - Retention graph

There is also the retention graph that demonstrates the percentage of users who return each day from Jan 26 to Mar 7. On day 4 retention has fallen to 80% and steadily goes down to 2%.

To have an audience, a specific application has to have a churn rate lower than a number of acquired customers [8]. Which makes customer retention an important factor of a game in defining the game's lifetime value and expanding the audience.

$$LTV = \Sigma\, retention \times Average\ Revenue\ Per\ Daily\ Active\ User$$

There is a list of things to do [8] to keep healthy retention within the game:

2.1.1. First session and first-time user experience

Obviously, there is no chance of keeping a customer, who received a negative experience during the first session and instantly left the game. Perfectly, the first experience shows the game's core design and features, so users are able to understand whether the idea matches their wishes. There could be negative outcomes of the unclear first session: Firstly, the idea meets the user's wishes in the long-term, but in the beginning, it is unclear, so the user will be lost. Secondly, the idea does not match a user's needs which is unclear from the first experience, as a result, the customer will both waste his time and end up unsatisfied. The first experience could be easily proven by the dynamic PCG, which could provide a special environment for a first session.

For this purpose was created a classical approach named **tutorial**, where the customer meets the game's world and explores the games' core features. The tutorial is a gate for both a game and a customer in long-term relationships.

In addition, there is an **instant games** approach provided by google play. The instant games feature provides an opportunity to actually play the first game without fully downloading. That is an innovative feature that solves crucial problems. Firstly, there is no need to download a game for a customer to

understand whether he or she needs it, thus, the choice is more conscious. Secondly, it is reducing the chance of losing customers during product presentation on the market because instant experience increases a chance to provide a positive impression.

2.1.2. Steady progression

Not so obvious as the previous, but still the straightforward fact is that all extra features and elements should not be opened too fast. So, the game process has to be gradual. If a person achieves at once 10 thousand points, logically, the next aim will be at least 10001 points, but if there is no extra progression left - the game becomes mundane.

With the aim of achieving steady growth, games provide various resources, levels, achievements and other extra items which makes customers interested in further progression. But, content creation is not a single method of achieving gradual advancement. For instance, some puzzle games attract people without having long-term progression plans or a variety of content, the only thing they do is a **restriction of session duration**. Thus, customers cannot achieve huge progress during a single session, otherwise, the user will become bored too fast because of an overwhelming amount of achievements.

2.1.3. Other

In addition, there are three more crucial elements that impact the game's retention and acquisition rate:

- Bonuses and gifts
- Keep reminding about the game. Customers need to be notified about updates, features and bonuses.

- Social interactions. Any interactions within people in the context of a game.

Despite things as UX, UI, bonuses, gifts and marketing are important for CRA, they will not be analysed in depth.

2.2. Summary

Customer retention and acquisition (CRA) are complex problems that could be affected by a variety of factors. There is a thing that could be done for a positive effect on CRA - improving the game's core logic in a way of adjusting it individually for every customer and every specific case.

First, the initial user experience would be tackled as a specific case of a content generation where the main aim of a session will be acquisition rather than a long-term effect. Eventually, a session could be bright and short rather than ordinary and moderate. Instant games together with tutorials are common techniques for providing first positive impressions and game's logic understanding.

Second, to retain customers, a game needs steady progression. Which could be implemented with a significant amount of content including levels, locations. As a result of this strategy, a player will be able to smoothly walk through the game.

Third, there is a solution for games that are running out of content: modification of core game logic in a way of controlling the duration of customers' sessions. For instance, in an endless game without any levels, it is possible to make sessions easy to play until a customer reaches the best result.

3. Game definition

Block Puzzle is a game, where the main mechanic is filling as many as possible cells with puzzles. The current game does not have levels or plot, the main idea is a smooth experience during the puzzles matching which makes the player relaxing during the process.

3.1. Project logical structure

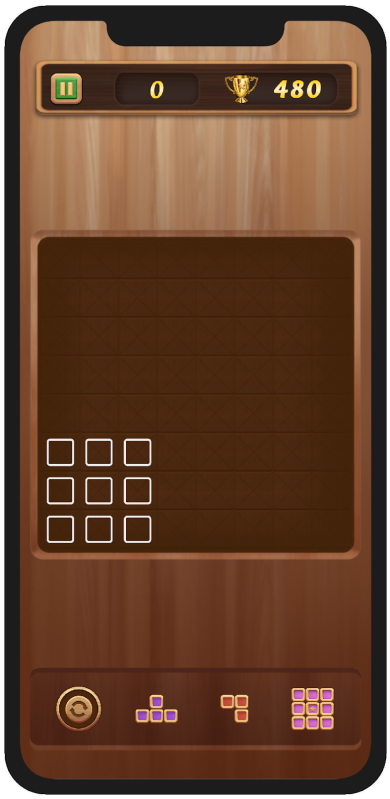

Figure 3.1.1. Main game screen

The game could be effectively decoupled on the two core components: puzzles row and puzzles grid. There are also such components as score, menu and so on, but they do not play a big role in the problem consideration.

3.1.1. Background Items Grid

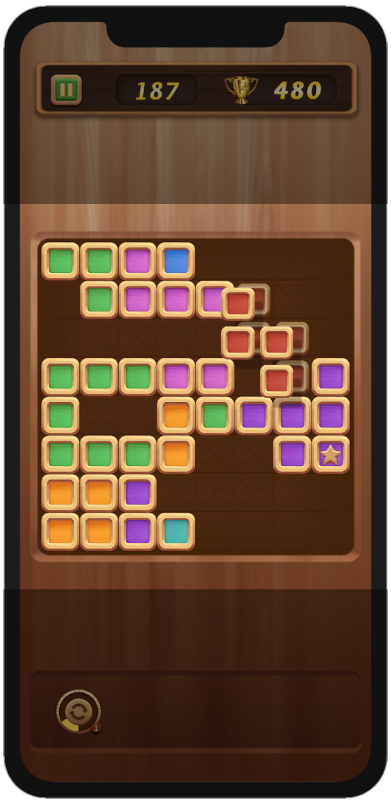

Figure 3.1.1.1 Background Items Grid

Background Items Grid is a game area for puzzles that have a size 8x8. Grid could be **filled** with puzzles and released by puzzle destruction effects. There are two types of **destruction**: simple and bonus. Simple destruction occurs when the grid has all the puzzles filled in a **row** or **column**, while bonus destruction is done in case the player failed the game and decided to use a bonus to resume it.

Bonus destruction cleans up an area where any new puzzle could be placed again.

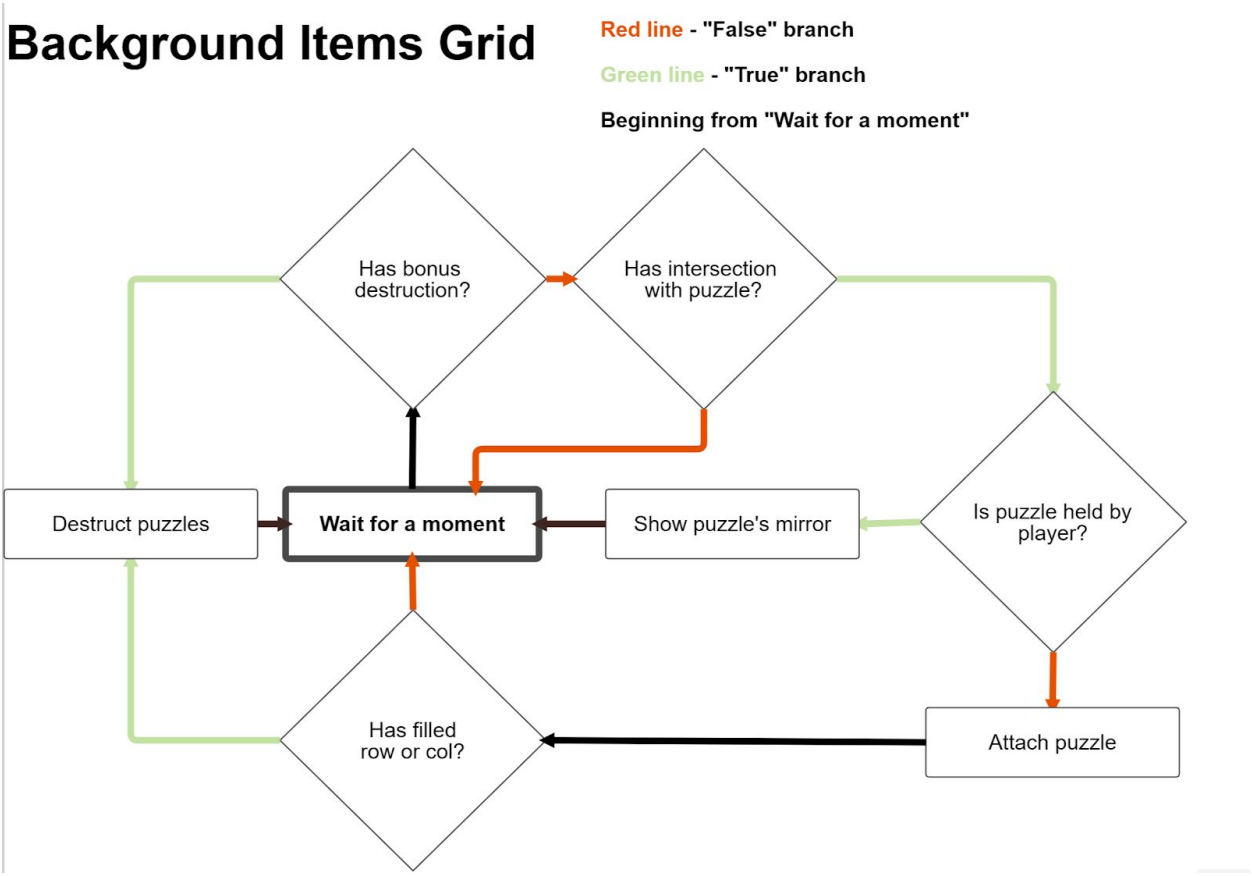

Figure 3.1.1.2. Background Items Grid block scheme

3.1.2. Puzzles row

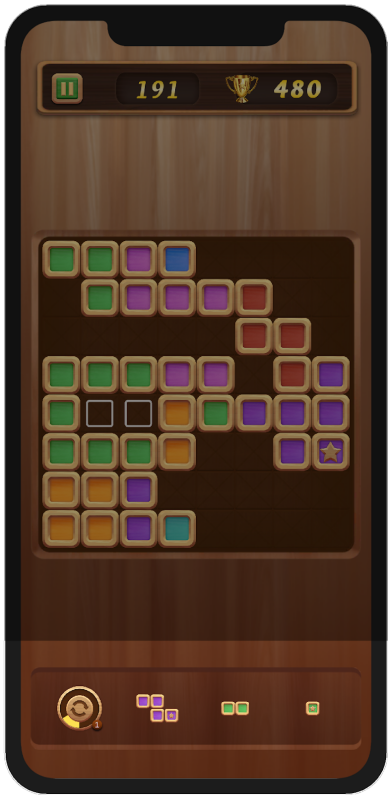

Figure 3.1.2.1 Puzzles row

Each turn player receives three puzzles, every of which has a specific type. Puzzles appear in the bottom **puzzles row** and could be dragged by a user and attached to the **background items grid**.

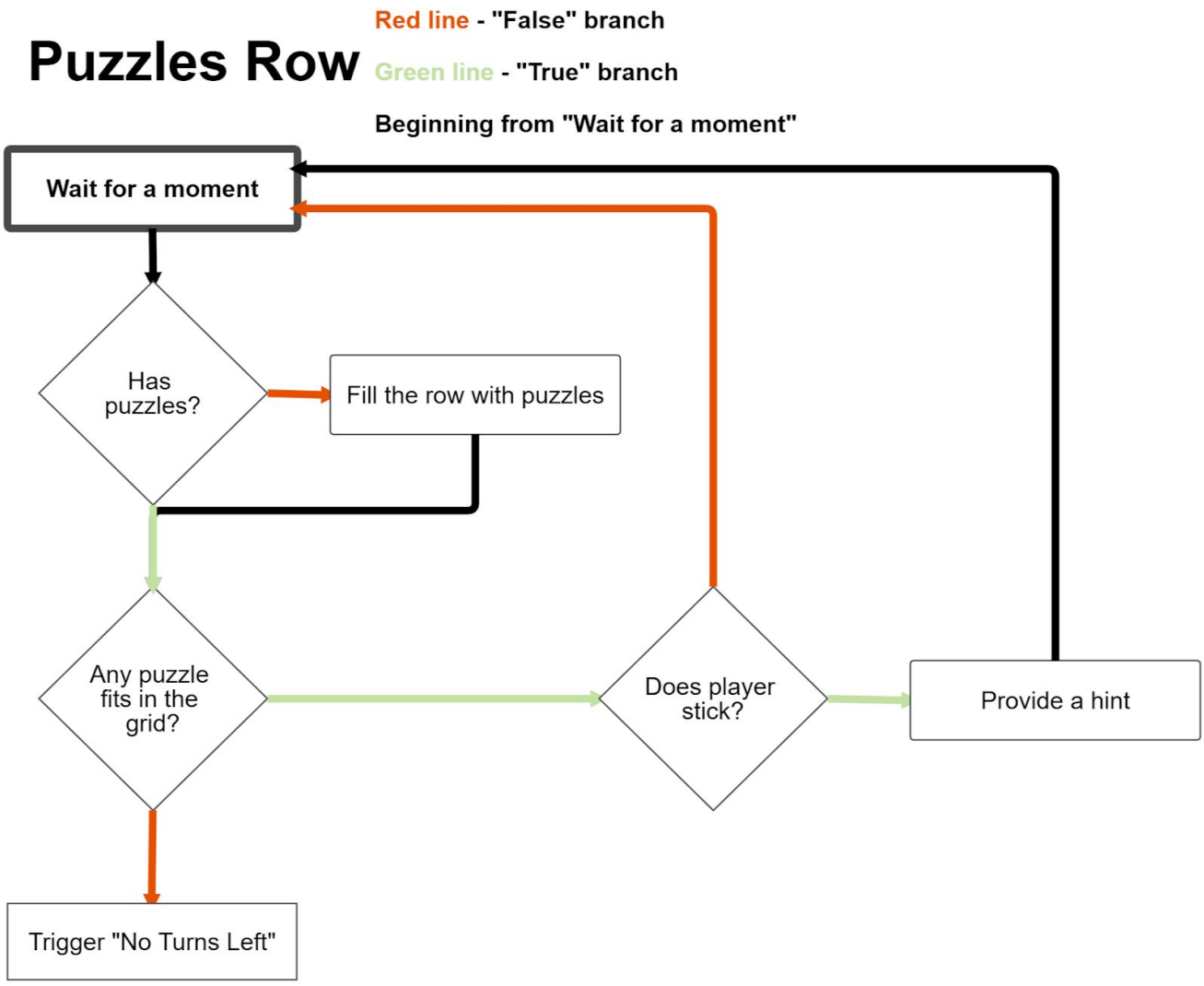

Figure 3.1.2.2 Puzzles row block scheme

3.1.3. Stars and rotations

Every puzzle is rotatable, rotation exactly means rotation of the puzzle at 90 degrees. There are no rotations at the beginning of the game, rotations could be earned by collecting stars during the puzzles' destruction. For a single rotation, the customer has to collect X stars and the user could have multiple rotations at a time.

3.1.4. Agent's role

The task of an agent is to make a customers' experience as enjoyable as possible with puzzle generation. In case of a random probability for each puzzle type, the game immediately loses its session control and most sessions become chaotic. As a result, the game will fail in its core logic due to the lack of feedback and an outcome on the customer's efforts that breaks all the motivation to play the game.

The duration of a game session heavily depends on the player's progress. A session remains as long as puzzles successfully match the grid and there is a place for new ones. Success requires two factors: suitable puzzles to match and a player's ability to match them, where the first condition is required for the second one. In case of chaotic puzzle generation, the player's ability will not play any role during the game, so the game is not flexible according to the player skills which have to be fixed. A workaround for this is a removal of the first condition, puzzles need to be suitable for most of the cases. As a result, when the first condition is satisfied, there are some puzzles to match, the only influence on the game session left for a player's skill. Eventually, players will always be able to achieve higher. Of course, every session filled with achievements could become mundane after some time and the scheme may be improved and modified somehow.

Thus, the task of the puzzle generation is laid on an agent which has to generate suitable puzzles at appropriate moments. Aim of an agent during the game is to generate appropriate puzzles that will be obviously suitable within the specific turn.

An agent has two roles: puzzles matching for the ability to provide hints and puzzle generation to empower the importance of player's skills in the game and reduce the influence of random factors. Puzzle matching is a short task on three turns which could be solved at the runtime algorithmically. But, puzzle generation is not such a trivial problem: First, puzzles ought to have a probability of being generated as evenly as possible to increase the variety of items for users. Second, the customer needs to feel all the features of the game, so during a session, he ought to earn and use as much rotation as possible too. Third, not all the customers have the same skill, so the generation ought to be adjusted for both newbies and pro gamers. And last, but not the least, customers have to be able to beat his best score within three games, so he will not sit at the single place.

Eventually, the solution of these cases lies in the right probability parameters for puzzles that have to be chosen after some experiments and control of these parameters to be relevant for the customer.

3.2. Project class diagram

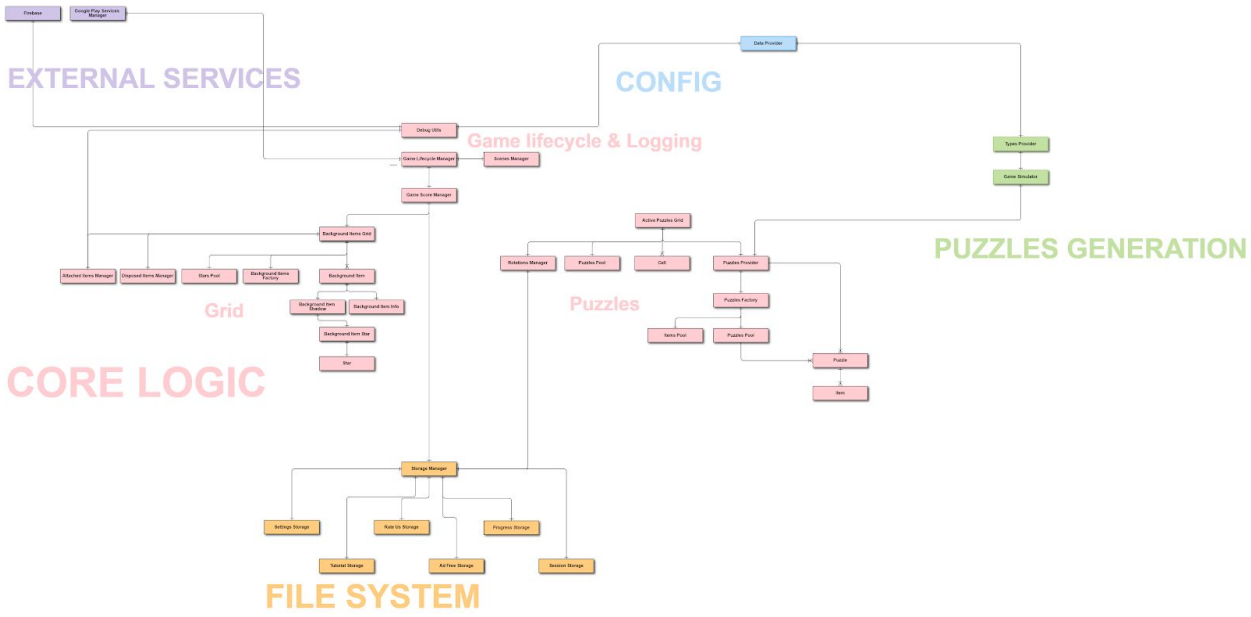

Figure 3.2.1 Class diagram

The actual game has 5 logic modules: Core, External Services, Config, Puzzles Generation, File system. Most of the classes and relationships that serve for UI purpose were suppressed for the diagram simplicity.

3.2.1 Core logic

As has been written before, the core module is divided into two visual (and functional at the same time) parts called **grid** and **puzzles** and the third functional one, **game lifecycle and logging**.

3.2.1.1 Background Items Grid

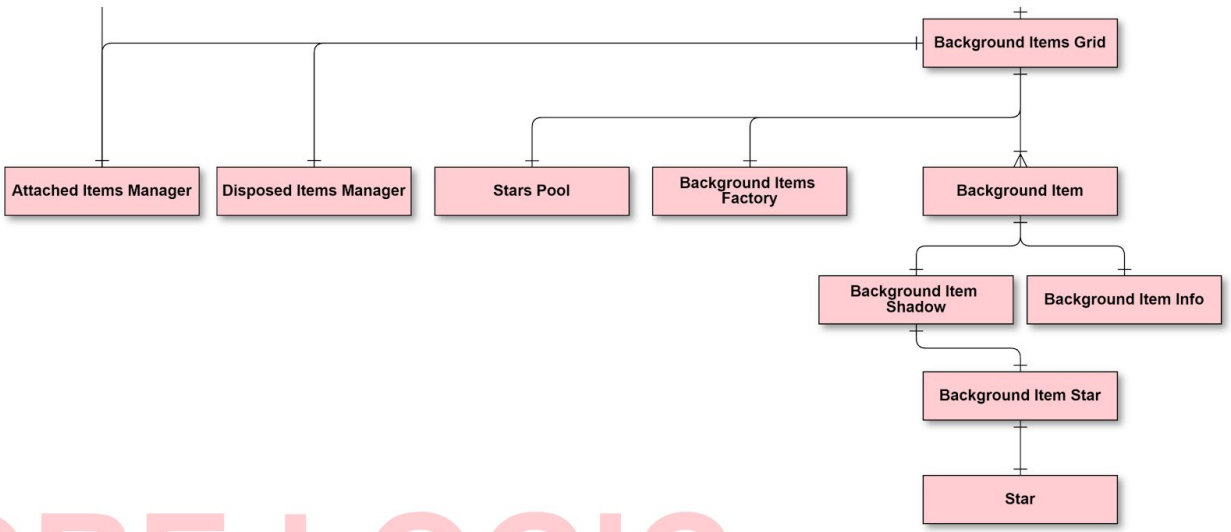

Figure 3.2.1.1.1. Grid functional diagram

Background items grid has a purpose of background items generation, attachment and destruction management. **Background item**s serve as a sensor of puzzle movement. Background items grid has **attached items manager** that is delegated to control grid state with the next logic:

1. *GC is a set of background items in the grid*
2. *GRC is a set of rows and cols items in the grid*
3. $\forall\ x,\ x \in GC,\ Filled(x) = true$ *if* x *has been filled with a puzzle*
4. $\forall\ x,\ x \in GRC,\ Filled(x) = true$ *if* $\forall\ item,\ item \in x\ Filled(item) = true$

   If every item or row/col is filled then row/col is filled
5. $\forall\ x,\ x \in GC,\ Distructible(x) = true$ *if* $x \in r,\ Filled(r) = true,\ r \in GRC$

   Coordinate is destructible if it is in the filled row.

Star serves as a star object for background items grid that is spawned only in the destruction case. Background item star is a static star for background item.

3.2.1.2 Puzzles row

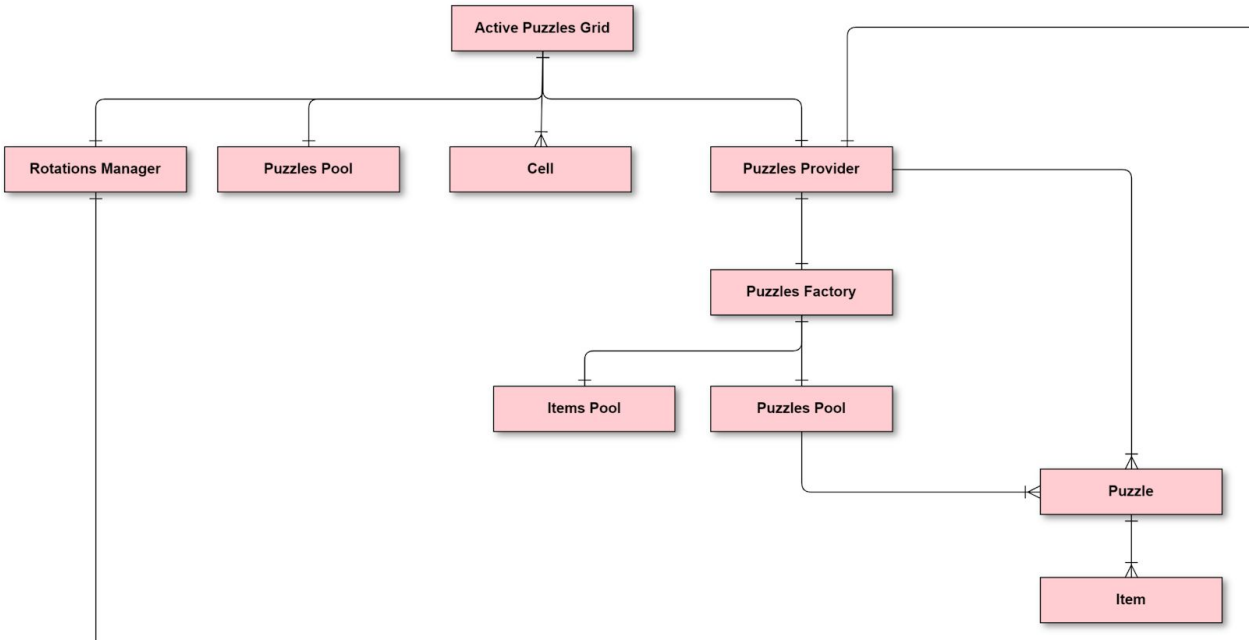

Figure 3.2.1.2.1. Puzzles row functional diagram

Active puzzles grid serves as the adapter between the customer and the puzzles generation module. Roughly speaking, everything it does - takes three puzzle types from the **puzzle provider** and presents them to the customer.

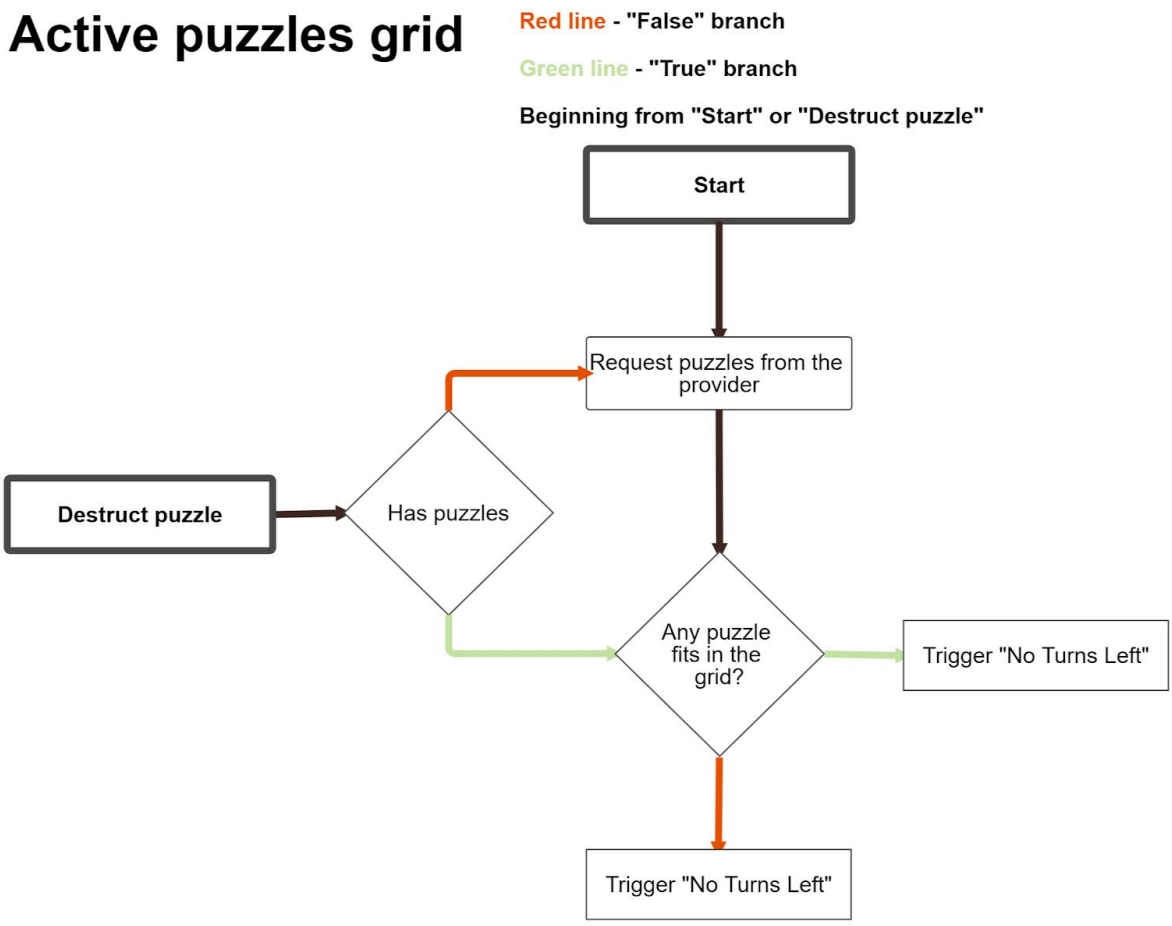

Figure 3.2.1.2.2 Active puzzles grid block scheme

Rotations manager is responsible for rotations providing, for receiving *S* stars it provides a single **rotation turn**. During the rotation turn the player is able to rotate generated puzzles any amount of time until one of the puzzles will be attached or the end of the game.

3.2.1.3 Game lifecycle and logging

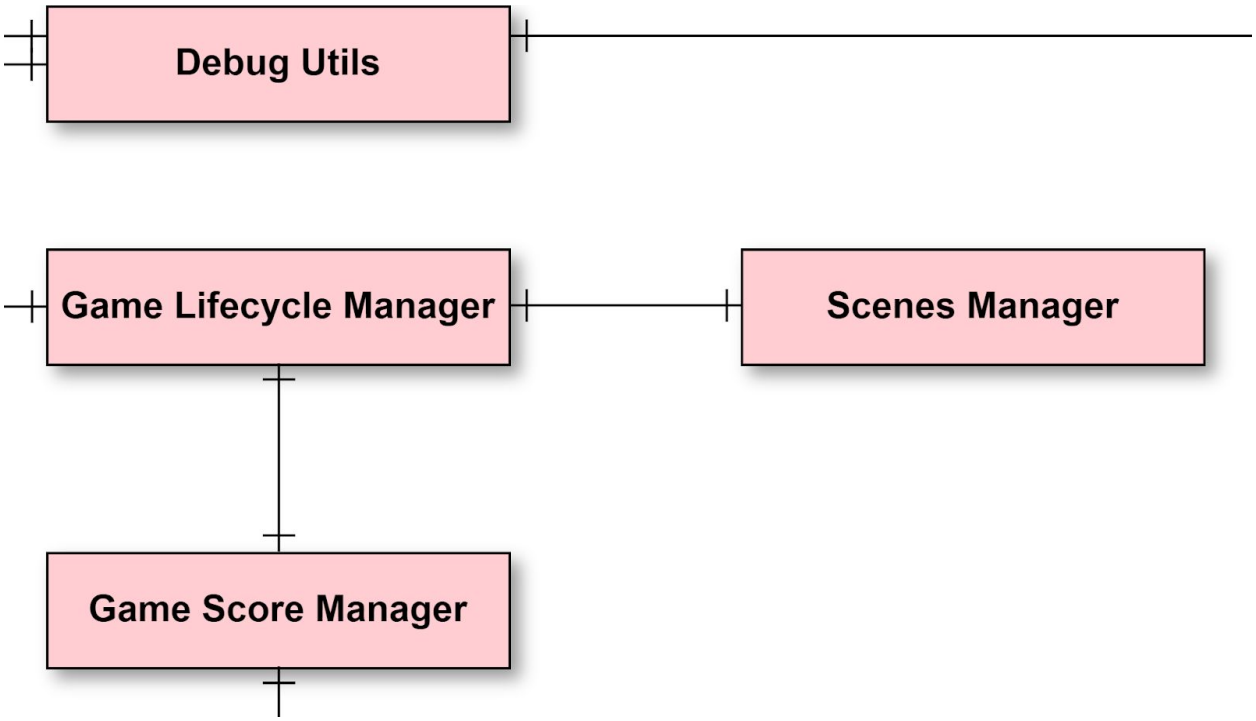

Figure 3.2.1.3.1. Game lifecycle and logging diagram

This module has trivial tasks such as notifying about records and errors to a cloud service, game lifecycle control.

3.2.2 Config

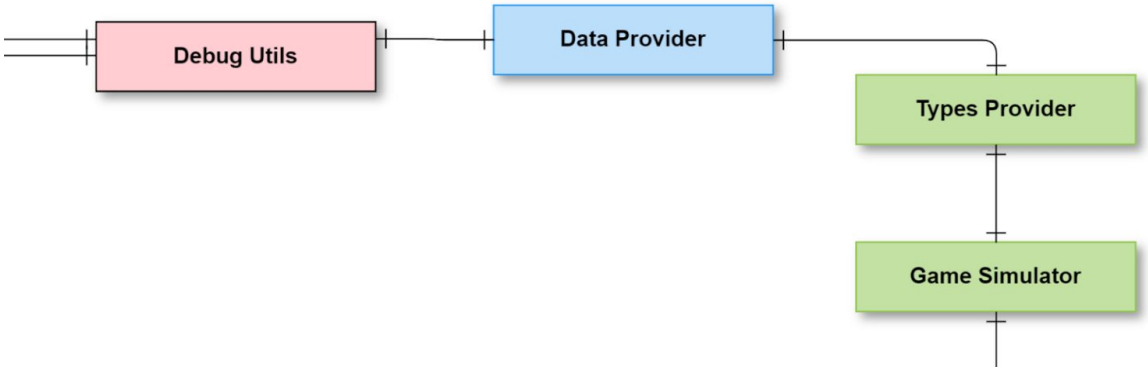

Figure 3.2.2.1. Config class diagram

Config is a set of rules that are predefined in the file before the build procedure. The most interesting detail in the current scope is that config provides puzzle types predefined in the file. Entire config module in the program represented by the **Data Provider**.

3.2.3 Puzzles generation

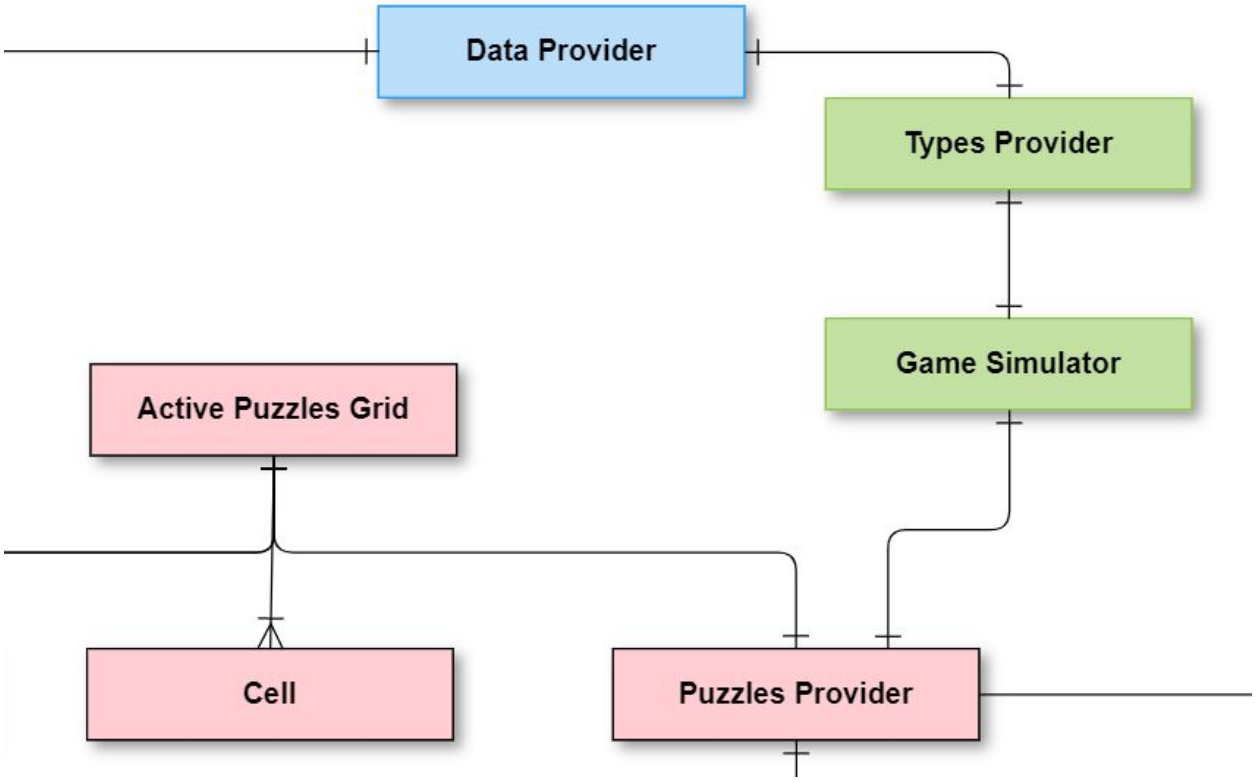

Figure 3.2.3.1. Puzzles generation class diagram

Puzzles generation module is represented by **Types Provider** and **Game Simulator** classes. **Puzzles Provider** retrieves puzzles from the Game Simulator on demand. Eventually, it is a task of the current work.

Game Simulator is a module that performs **procedural content generation** by simulating a three-turns in depth game with puzzles provided by Types Provider.

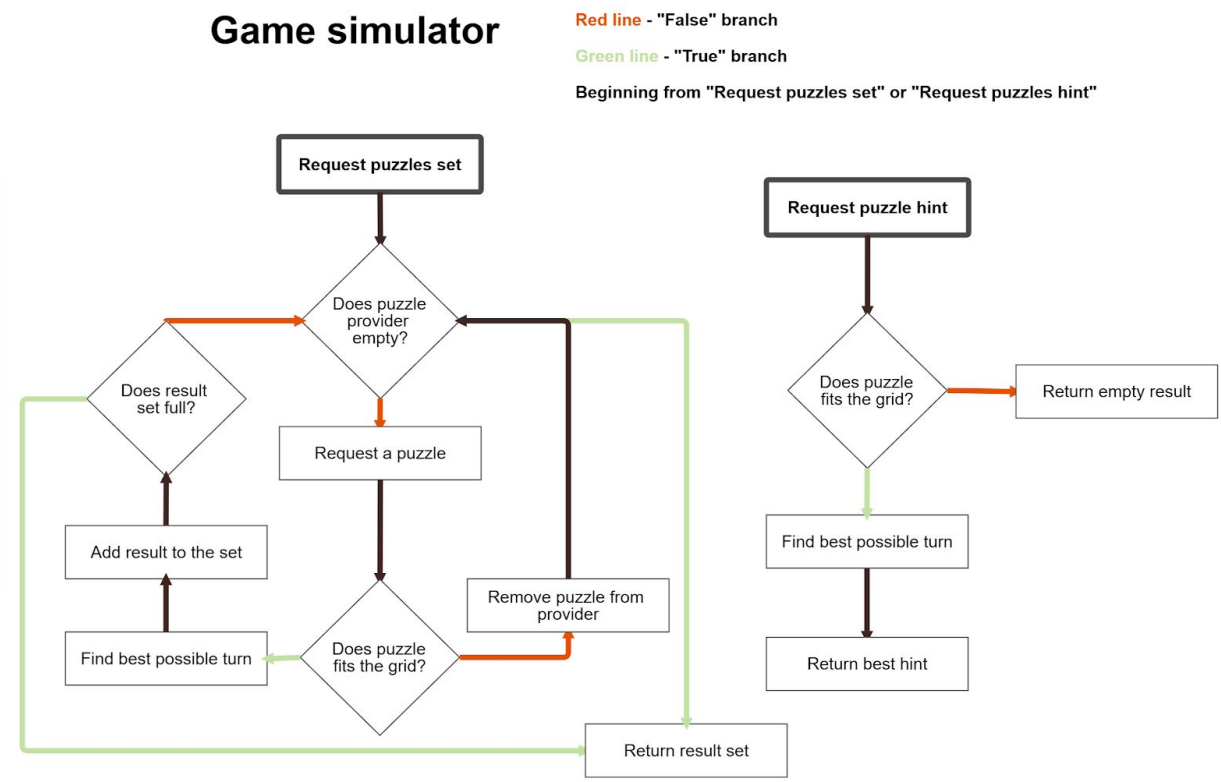

Figure 3.2.3.1. Game simulator block scheme

To add up to a scheme, "Find best possible turn" is a search algorithm that finds all possible matches on the board and picks up a one with the best possible score. Score is defined by the following formula (some rules are taken by 3.2.1.1 chapter):

1. PC *is a set of items in a puzzle*
2. $\forall x,\ x \in PC,\ F(x) = true$ *if* x *is fits in the grid*
3. $\forall x,\ y,\ x \in PC,\ y \in GC,\ Adj(x,\ y) = true$
   if y *is vertically or horizontally adjacent to the* x
4. $score = \Sigma\,(F(x) + \Sigma\,Adj(x)),$ *where* $x \in PC$

Procedural content generation occurs exactly in the game simulator. The most of the balance lies in the **"Request a puzzle"** step, where **Types Provider** provides a type for puzzle with a predefined probability. These parameters ought to

be learned using reinforcement learning with an intellectual agent with a learning environment.

4. Reinforcement learning

The tuning of probability parameters demands a big amount of experiments and sessions analysis in case it is provided by people, but reinforcement learning makes this process much faster and precisive and below will be explained why.

4.1. Introduction

"Reinforcement learning is an area of Artificial Intelligence; it has emerged as an effective tool towards building artificially intelligent systems and solving sequential decision making problems. Reinforcement Learning has achieved many impressive breakthroughs in recent years and it was able to surpass human level in many fields; it is able to play and win various games. Historically, reinforcement learning was efficient in solving some control system problems. Nowadays, it has a growing range of applications." (Hammoudeh, Ahmad. (2018). A Concise Introduction to Reinforcement Learning. 10.13140/RG.2.2.31027.53285.) [9]

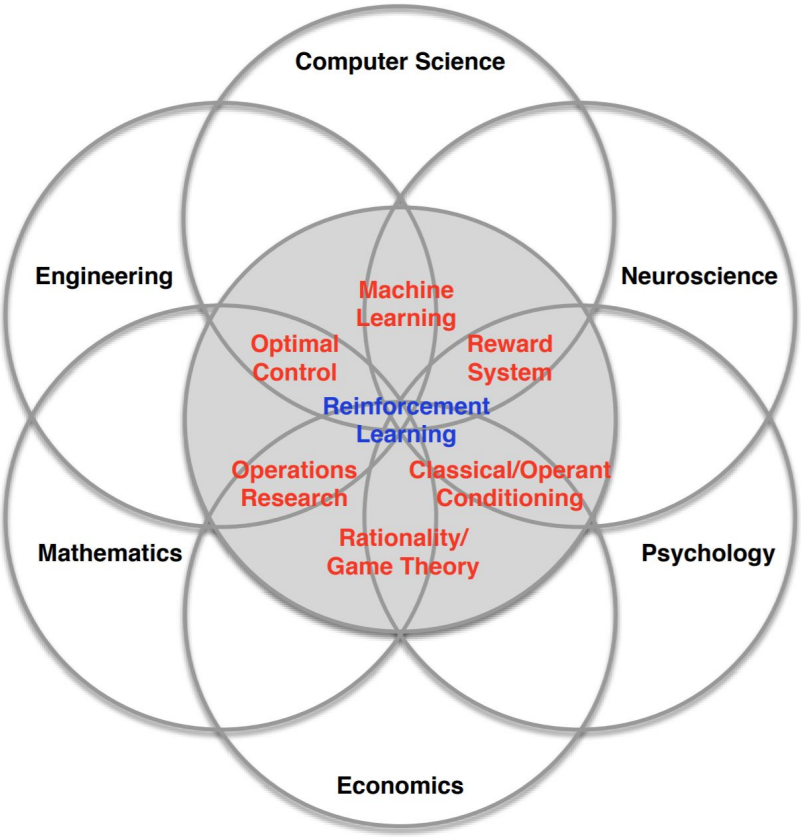

Figure 4.1.1. Many Faces of Reinforcement Learning [10]

For the current problem, reinforcement learning is a convenient type of learning that does not require a supervisor, but it differs from unsupervised learning that serves for finding a pattern within a collection of unstructured data. Supervised and unsupervised learning terms do not fully cover the classification of machine learning paradigms. Reinforcement learning is executing a series of experiments with an aim of reward maximization, it does not find a latent structure within a data. Reinforcement learning is considered to be an extra machine learning paradigm apart from supervised and unsupervised machine learning paradigms and other paradigms too. [11]

To achieve the best reward level an agent relies on steps from the past he remembered to be effective. This is called the exploitation of existing steps to achieve a reward. But, to have such steps in memory there is a need to make some decisions, so actions made by the agent will be remembered and will be able to exploit that called exploration. To obtain a reward the agent has to exploit steps it already grasped and to have steps to exploit it has to explore them first. Eventually, the agent should try a lot of actions to find the best way to earn a reward. By the way, the exploration-exploitation dilemma does not arise in both unsupervised and supervised learning types. [11]

Reinforcement learning starts with an interactive reward-seeking agent. A reinforcement learning agent always has an explicit goal and an uncertain environment where it will find ways to interact to earn a reward.

4.2. Elements

Apart from an agent and an environment, reinforcement learning has four elements called: a policy, a reward signal, a value function and model which is an optional one. [11] This chapter will discuss theoretical aspects of the main elements of reinforcement learning based on explored literature and papers.

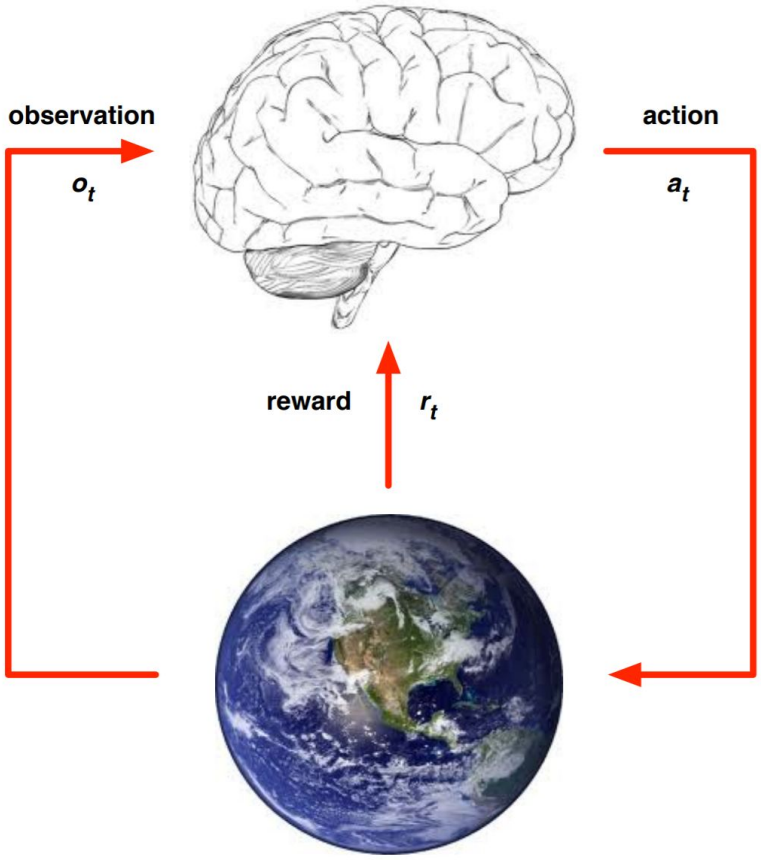

Figure 4.2.1 Agent and Environment [10]

At each step t the **agent** executes action a_t, receives observation o_t and receives scalar reward r_t. The **environment** receives action a_t, emits observation o_{t+1} and sends out scalar reward r_{t+1}.

4.2.1. Policy

A policy is a definition of how a learning agent acts at a given moment. Policy could be considered as a mapping from incoming environments' states to concrete actions of the agent in these conditions. On the one hand, policy could be a simple function or a lookup table that directs from a state to an action. But, on the other hand, policy easily could involve a computation such as a search progress.

The policy is the core of the reinforcement learning agent because of the reason it is the sufficient factor to determine the agent's behaviour. [11]

4.2.2. Reward signal

A reinforcement learning problem's goal is called reward signal. Each time period reinforcement learning agent receives a number from the environment called reward. The aim of the reward signal is to notify the agent about good or bad actions in a way of sending positive or negative reward numbers respectively. Every action agent aims to maximize the reward it receives during the long run. For instance, an agent's reward could be compared to biological experience of pleasure and pain that are immediate in defining a problem during the learning process. [11]

The reward is sent to an agent at any time based on some either agent's or environmental parameters. This process cannot be suppressed by the agent or influenced directly. The only way to earn reward is to provide a successful action or to influence the environment somehow. The reward is a direct way of altering the policy. The action followed by low reward will be changed to another one in the future. [11]

4.2.3. Value function

While the reward signal provides a short-term effect on the agent, a value function defines an outcome for a long run. "Value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state" [11]. Value function provides long-term strategy from the current state by looking at states the agent is likely to follow and accounting the reward available at these

states. For instance, controlled by a value function an agent will prefer path 1->2->3->100 rather than 20->20->20->20. Projecting a value function on the human world, it is rather someones' annual plan of how much could be acquired following some plan. [11]

4.2.4. Model

The last element of the reinforcement learning system is an environmental model. The model allows agents to make predictions about how the environment will behave. For example, knowing current state and action, next both action and state could be predicted by the model. Common case of model usage is planning which means considering possible future situations before they are actually experienced. Reinforcement learning problems solving methods with model and planning usage called model-based, while opposite methods to them called model-free methods that are pure trial-and-error learners. [11]

An accurate example of model-based reinforcement learning is the chess or checkers game. Almost everyone during the game thinks about the opponent's possible actions after the move which is direct model analysis. So, model-based agents are the same players that analyse the model. For instance, the agent could analyse the game board and all possible outcomes after the moves and as a result choose the most successful one.

5. Unity platform as a learning environment

5.1. Introduction

The agent needs an environment for the reinforcement learning process. Unity ML-Agents Toolkit [12] is an open source toolkit for creating and interacting with simulation environments using Unity platform. Unity toolkit enables the development of learning environments which are rich in sensory and physical complexity, provide compelling cognitive challenges, and support dynamic multi-agent interaction. [13]

In recent years, during the variety of Reinforcement learning researches, learning environments played an important role in this development. For instance, Arcade Learning Environment [14] that is a platform and methodology for evaluating the development of general, domain-independent AI technology or VizDoom [15] which is a Doom-based AI Research Platform for Reinforcement Learning from Raw Visual Information and others. Creation of high-quality environments for machine learning algorithms is time-expensive and requires specialised domain knowledge, so creation of an environment for a specific case rather than using existing one is quite inefficient in terms of resource usage. These models not only served as sufficient learning environments, but also as environments for models that would be deployed in the real world later on that makes them quite important in the reinforcement learning history. [13]

Current chapter will provide discovered properties of a simulation platform, and present the Unity platform itself.

5.2. Environment properties

Models are able to solve increasingly hard tasks that require an increasing complexity of simulated environments in order to algorithms being explored. Defined complexity keys of the environment are believed to be: sensory, physical, cognitive and social. [13]

Sensory complexity - This complexity is required for a variety of real-world decision making problems, for instance, self-driving cars, household robots. [13]

Physical complexity - Plenty of researchers interested in solving problems using AI not only using sensory analysis, but also in a way of a complex interaction with an environment. [17] The need of complex interactions often comes in a real-world problem. Usually, the main goal of such problems is to transfer the experience from the environment to the real-world. This approach is quite useful for robotic applications. [13]

Cognitive complexity - The next complexity is a cognitive which is also called combinatorial complexity. A sufficient example of such an environment is Go game that has served as a learning environment without having both complex visual and physical interactions. Cognitive complex tasks are able to present hallmarks of human intelligence and hierarchical structure. In addition, tasks may be presented in a consecutive manner, so independent examples without a context of previous cases (except from the first example) are not valid. Cognitive tasks often occur in the real world with people. [13]

Social complexity - The outstanding progress in acquiring complex skills by learning in mammals believed to be achieved due to communication skills. [18] Development of social skills and relations within a group of agents is a particular

interest in a variety of researches in the AI field. In addition, the ability of communication is a key feature of the ecosystem's development in the world, which would be desirable to simulate. An environment that allows a study of communication and social relationships should provide a multi-agent communication framework for communication between agents from a similar group and from different distributions as well. [13]

5.3. Simulation properties

While environment properties define a sufficient environment for the learning process, there are some prerequisites for the simulation process. Simulation environments must be easily controlled by a customer who is usually a researcher. [13]

First, is a fast and distributed simulation. Depending on the method used, modern machine learning algorithms require billions of pieces of data in order to find an optimal solution. One of the most important advantages of the environmental simulation is that environmental simulation is usually faster than a physical one. Therefore, environmental simulation has to be as fast as possible while providing a flow of experiments. In addition to speed, simulation ideally should be runnable in parallel, so it provides a significantly greater throughput of data compared to the physical world. The faster algorithms could be trained, the greater speed of receiving outcomes from an experiment. [13]

Second property is a flexible control. As any software, simulators should be convenient to use and provide flexible control to the researcher making opportunities of environmental simulation as far as possible to the real-world one. Using a simulator as a black box could be enough for some experiments, but there

are some approaches such as **Curriculum Learning**, **Domain Randomization** or adaptive task selection. In these approaches dynamic feedback between the training process and the agents is crucial. [13]

Curriculum Learning is a learning process where an agent receives a sequence of tasks for invocation to achieve a final result. This method was used to achieve approximately human-level performance in a VizDoom competition. It is assumed that to achieve a success in such approach there is a need to influence the simulation. [13]

As an example of curriculum learning could be considered a task of training a penguin agent to catch the fish in a lake and feed his baby. This task has been already solved in a blog called "Immersive limit". [19]

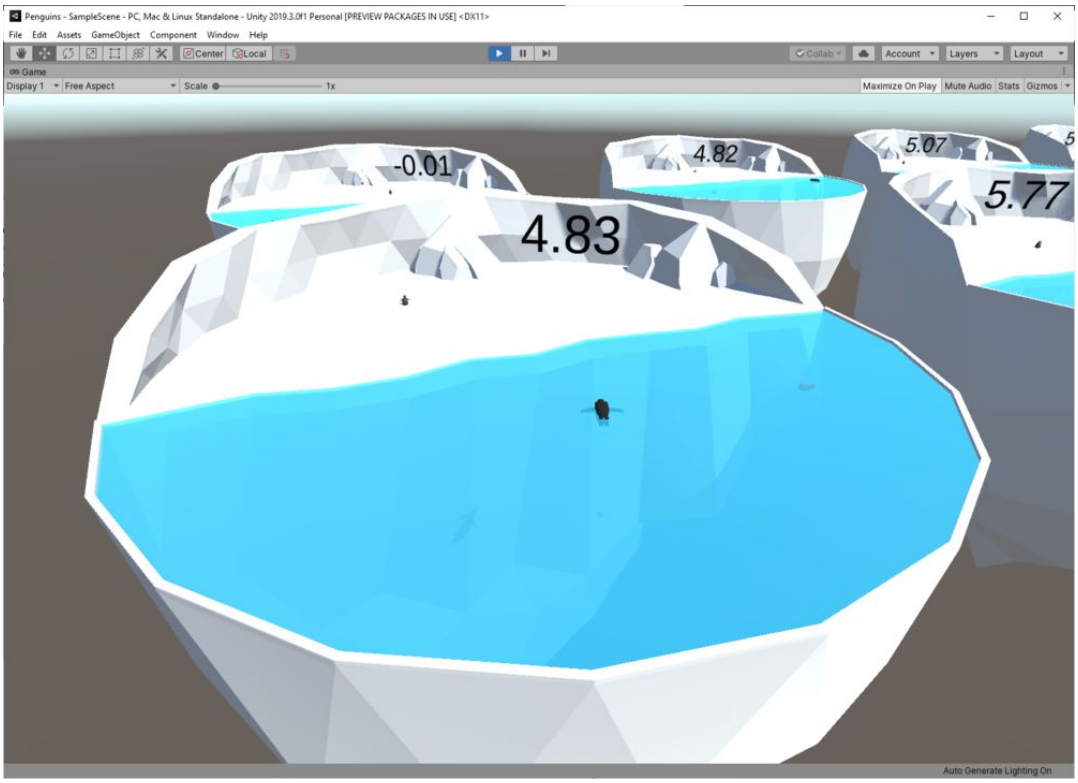

Figure 5.1 - Penguins collecting fish using neural networks to make decisions [19]

This concrete task has been solved by a curriculum learning approach. The final aim is to feed a baby as fast as possible. But, the first task looks like "Feed a baby as fast as possible by considering the feed range as 6 meters and do it until reward value X will be met". The second task considers the range as 4 meters and reward value Y. And so forth until the end of curriculum with the final task where the range is range of penguins collision with maximum possible reward.

During the simulation there are parameters to configure in the curriculum such as measure for lesson from the curriculum finishing, threshold in the reward for each level, minimum lessons length before going to the next lesson, so a chance of random result is reduced and fish speed together with radius are also configured. This example shows how convenient the learning process with curriculum could be in terms of simulation controlling.

Another technique is a **Domain Randomization** that requires flexible control over the simulator. This approach learns from an environment during the simulation, so the experience could be projected on a real world. The approach will work if a virtual environment is sufficient in terms of functional elements to cover the real world. This is especially important in case an agent's behaviour depends on visual components of the environment. Domain Randomization usually means modifying of textures, lighting, physics, and object placement within a scene. Without this approach trained models usually end up having a "reality gap" and perform bad in the real world. [13]

5.4. Unity Editor

Unity Editor is a graphical user interface used to create the content for 2D, 3D, AR, VR experiences.

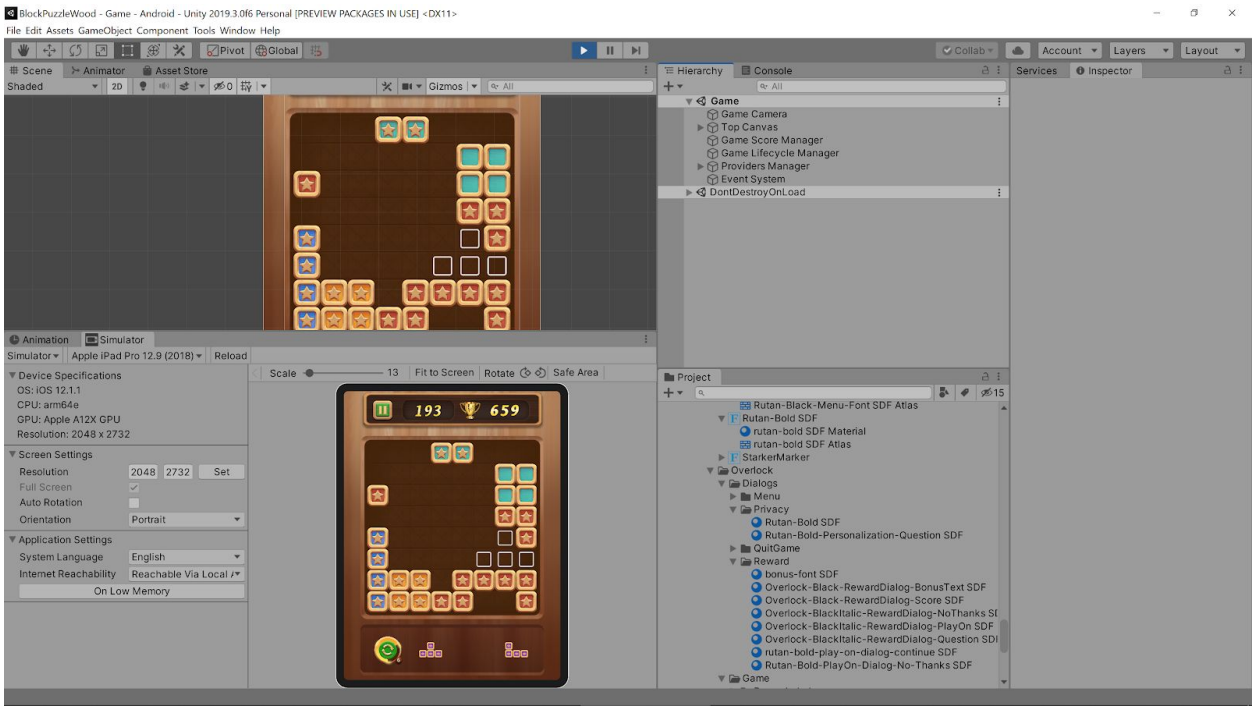

Figure 5.4.1. Unity Editor

The Unity Editor together with external services provides next extra opportunities for AI research [13]:

1. Custom scenes creation. Unity provides an ability for creation of scenes with any complexity, from a race environment for cars to a 2D environment for games.
2. Record demonstrations. Unity Editors' play mode provides an ability to demonstrate any simulation flow on the scene.
3. Provide demonstrations on a wide range of devices. The Unity Editor is compatible with more than 20 platforms, so a project could be runned on a variety of devices.

5.5. ML-Agents Core Toolkit

The ML-Agents Core Toolkit is a two-part module that provides all necessary technologies for the simulation environment. It has two components: ML-Agents SDK that is a part of Unity Editor written in C# and a Python package that communicates with the environment in Unity using SDK. [13]

5.5.1. ML-Agents SDK

ML-Agents SDK makes it possible to make a Unity's Scene into a learning environment using three SDK entities: Agent, Brain and Academy. [13]

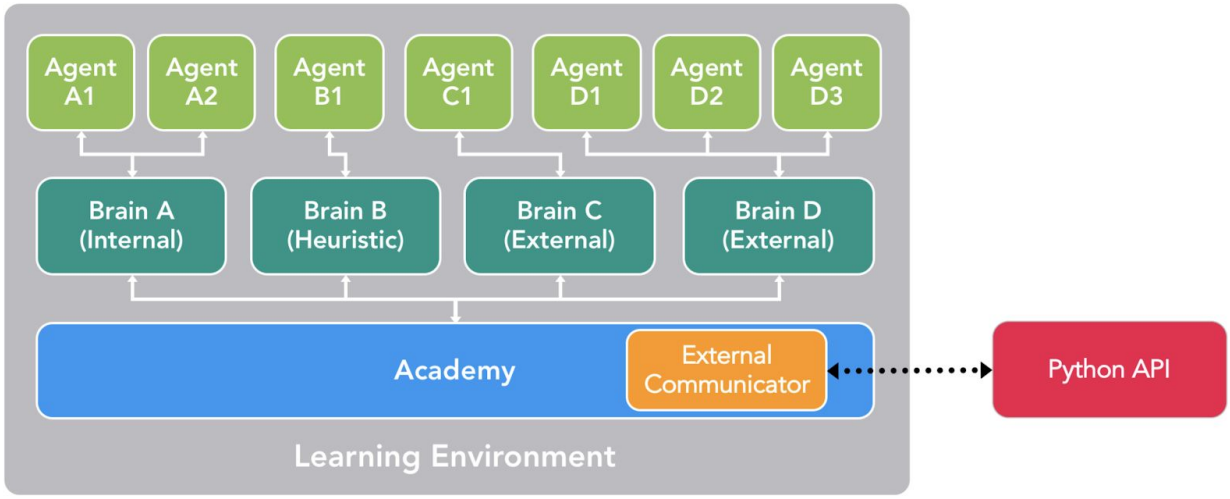

Figure 5.5.1.1. Learning Environment [13]

The Learning Environment contains Agents, Brains and an Academy. Agents are responsible for collecting observations and taking actions. Brains are responsible for providing a policy for agents it has. The academy is responsible for entire environment simulation management. [13]

The agent component is used to indicate a GameObject as an Agent who is responsible for observing the game world. Each Agent linked to a single Brain at a

time. There is possible to create a multi-agent environment by using more than a single agent in the environment by simple objects' duplication. [13]

Brain components make decisions for all their linked agents by providing them a policy. Brains has a definition of an area observation, so the policy will be valid in the specific observed environment. Agent is able to be linked to a brain only by having observations and actions defined. Brain's decision could be concluded in several ways including player input, scripts, internally embedded neural network models or via interaction with Python API. Thence, policy providing respectively could be referred as Player, Heuristic, Internal and External. Multiple brains makes it possible to take a place of multiple policies. [13]

The Academy module is responsible for simulation step tracking, environment reset functionality, target simulation speed and frame rate setting. In addition, the Academy could define reset parameters that could be used to change configuration of the environment during runtime. [13]

Once an agent is placed in the scene then the environment could be enabled together with a Markov Decision Process (MDP) or Partially Observable MDP (POMDP). Where observations, reward functions and scoring are placed in the agent. Moreover, scoring could be provided by the environment. [13]

Observations could be provided in two ways: As ray-casts, auditory signals, Agent position and so on for complex environments or via the numerical vector for vector environments. Actions could be sent by discrete signals like arrays of integers or as continuous variables (floating point numbers array). [13]

Agents could ask a brain for a decision at a fixed or dynamic interval. The reward function could be modified at runtime as well. Simulation could be finished in several ways: for a single agent or for a whole environment. Besides, it happens

after reaching max steps count predefined in Unity Editor. And last, but not the least, it is possible to configure environmental reset variables from the Python API.

5.5.2 Python module

The Python package (https://pypi.org/project/mlagents/) contains code that could be used to launch and interact with Unity executables and with Unity Editor as well. Communication between Unity and Python is made using gRPC. [13]

6. Learning process

6.1. Learning environment

For the purpose of agent learning there is a need for a specific Unity environment, where the agent will be able to do observations and receive a reward based on its actions. Thus, only sufficient for a simulation architecture elements will remain.

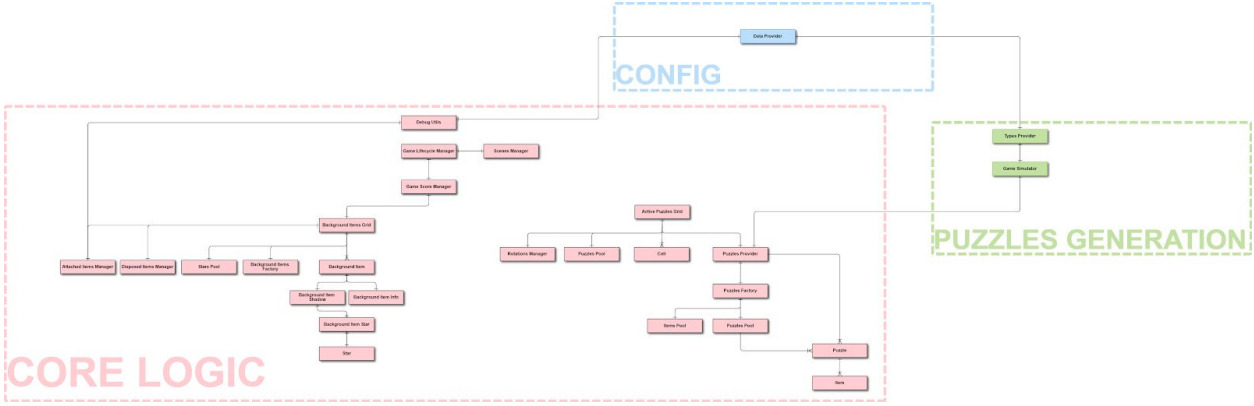

Figure 6.1.1. Learning Environment class diagram

There are no more external API and File system modules, in addition, UI modules were extremely cut off as well.

For a simulation purpose there is a need to define two main simulation elements: the learning area and the agent.

The learning environment is an object that knows and is able to reset every stateful object on the scene such as **BackgroundItemsGrid**, **RotationsManager**, **PuzzlesProvider** and **ScoreManager**.

Puzzles Area (Some code was removed for simplicity purpose):

```
public class PuzzlesArea : Area
{
  [SerializeField]   private BackgroundItemsGrid backgroundItemsGrid;
  [SerializeField]   private RotationsManager rotationsManager;
  [SerializeField]   private PuzzlesProvider puzzlesProvider;
  [SerializeField]   private AreaScoreManager scoreManager;
  private void Start() => ResetArea();
  public override void ResetArea() {
    if (backgroundItemsGrid.IsPrepared) backgroundItemsGrid.Reset();
    if (rotationsManager.IsPrepared) rotationsManager.Reset();
    if (puzzlesProvider.IsPrepared) puzzlesProvider.Reset();
    if (scoreManager.IsPrepared) scoreManager.Reset();
  }
}
```

6.2. Intellectual agent

The intellectual agent has to receive an **action** from the brain, make a **request** for an action or a decision. Obviously, it has to receive a **reward signal**.

And last, but not the least it has to receive **observations**. The agent is a PuzzleTypesProvider that operates in a vector space.

The action is a vector with length that equals the number of puzzle types to choose. Every value is a probability for concrete type to be chosen that normalizes then with a formula: $typeProbability_i = \frac{p_i}{\sum_{j<|P|}^{j} p_j}$ where P is a set of probabilities for a puzzle type.

The reward is defined by the next rules:

1. GCD *is a set of disposed items in the grid*.
2. GCA *is a set of attached items in the grid*.
3. tf *is a turn fee*.
4. P *is a set of normalized probabilities for puzzles types*.
5. mf *is a median fee*. $mf = \frac{median(P) - a}{|P|} * b$, *where a and b are parametric*.
6. $mr = GridSize^2 + 2RewardPerDisposedRow * 2GridSize + Max(PuzzleSize)^2$

   Where mr is a maximum reward.
7. $reward = \frac{(\sum_{i<|GCA|}^{i} x_i + \sum_{j<|GCD|}^{j} y_j) - (tf + mf)}{mr}$

6.3. Results

This chapter will introduce a result of the learning process.

Thus, the best score achieved by an agent during learning sessions was 1154. The reward is 313 and has a growing passion. To introduce the last result, the

puzzle types, there is a need to explain the format. Puzzle type could be represented by an id, coordinates and color.

Coordinates are represented in a specific way of filling a 5x5 grid with symbols, where p - mean puzzle, s - star, e - empty space. For instance,

- **Id**: 22
Coordinates: |
eeeee
eeeee
pppee
pspee
pppee
Color: Pink

in the game looks in this way:

Figure 6.3.1. puzzle picture

All the puzzle types are provided in the Listing 1. And the resulting vector of probabilities looks like this: [0.041(6), 0.041(6)] that is means equal probability for all the puzzles.

This result seems to be the best for an agent due to the reason that the zero distance between the puzzle types' probabilities paid well in terms of reward. This result is applicable and shows how much might be earned in a straightforward way using given puzzle types and parameters by the search algorithm, 1000+ points for a single flow is more than good due to the possibilities of a game simulator that could boost this result providing more applicable puzzles in specific situations.

7. Conclusion

Procedural content generation is an instrument for enhancing customers' satisfaction in a way of controlling game balancing. Game balancing could be dynamic or progress-sensitive, progress-sensitive game balancing is more applicable for customers satisfaction, but requires a learning process. Unity platform is a multifunctional learning environment that allows to create both physical and vector-based environments for reinforcement learning.

7.1. Practical results

Within this work has been accomplished

- Review of dynamic and progress-sensitive game balancing
- Analysis of customer retention and acquisition
- Discussion of customers retention and acquisition problems in the context of balancing in procedural content generation
- Deconstruction of the balancing problem for the specific block puzzle game
- Searching of benefits of the PCG balance in terms of customers retention and acquisition
- Definition of procedural puzzles generation algorithm

- Definition of reinforcement learning problem for configuration of probabilities for the PCG algorithm
- Revision of the reinforcement learning elements
- Review of the Unity platform as a learning environment
- Implementation of the learning environment and the intellectual agent for the PCG balance problem solution
- Listing of the learning process results
- Analysis of the algorithm performance
    - Average session score duration without the PCG algorithm is nearly 400 points that lasts nearly three minutes
    - Average session score with the PCG algorithm is 1200 points that lasts nearly eight minutes

References

1. https://www.researchgate.net/publication/331717755_A_Short_Introduction_to_Procedural_Content_Generation_Algorithms_for_Videogames
2. https://www.researchgate.net/publication/220978366_Dynamic_Game_Balancing_An_Evaluation_of_User_Satisfaction
3. Robin Hunicke; V. Chapman (2004). "AI for Dynamic Difficulty Adjustment in Games". Challenges in Game Artificial Intelligence AAAI Workshop. San Jose. pp. 91–96.
4. https://link.springer.com/article/10.1007/s10994-006-6205-6#Bib1
5. https://blog.count.ly/how-to-simply-calculate-user-retention-and-loyalty-7c272ee73edd#.gfmnvv1up
6. https://play.google.com/store/apps/details?id=com.bellkross&hl=en
7. https://middlesexconsulting.com/churn-rate-retention-rate-metrics/
8. https://thetool.io/2018/user-retention-mobile-apps-games
9. https://www.researchgate.net/publication/323178749_A_Concise_Introduction_to_Reinforcement_Learning
10. D. Silver, “Deep reinforcement learning,” in International Conference on Machine Learning (ICML), 2016.
11. R. S. Sutton and A. G. Barto, Reinforcement learning : an introduction, 2nd ed. Cambridge, MA: Mit Press, 2017.
12. https://github.com/Unity-Technologies/ml-agents
13. https://www.researchgate.net/publication/327570403_Unity_A_General_Platform_for_Intelligent_Agents
14. Bellemare, M. G. et al. “The Arcade Learning Environment: An Evaluation Platform for General Agents.” Journal of Artificial Intelligence Research 47 (2013): 253–279. Crossref. Web.

15. https://arxiv.org/abs/1605.02097
16. E. Todorov, T. Erez and Y. Tassa, "MuJoCo: A physics engine for model-based control," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, 2012, pp. 5026-5033. doi: 10.1109/IROS.2012.6386109
17. A. Bicchi and V. Kumar, "Robotic grasping and contact: a review," Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), San Francisco, CA, USA, 2000, pp. 348-353 vol.1.
18. https://www.researchgate.net/publication/24357265_Primate_Vocalization_Gesture_and_the_Evolution_of_Human_Language
19. https://www.immersivelimit.com/tutorials/reinforcement-learning-penguins-part-1-unity-ml-agents

Listing 1

```
- Id: 0
 Coordinates: |
   eeeee
   eeeee
   eeeee
   eeeee
   seeee
 Color: Green
- Id: 1
 Coordinates: |
   eeeee
   eeeee
   eeeee
   eeeee
   ppeee
 Color: Green
- Id: 2
 Coordinates: |
   eeeee
   eeeee
   eeeee
   eeeee
   pppee
 Color: Green
- Id: 3
 Coordinates: |
   eeeee
   eeeee
   eeeee
   eeeee
   pppse
 Color: Blue
- Id: 4
 Coordinates: |
   eeeee
   eeeee
   eeeee
   peeee
   peeee
 Color: Green
- Id: 5
 Coordinates: |
   eeeee
   eeeee
   peeee
   peeee
   peeee
 Color: Green
- Id: 6
 Coordinates: |
   eeeee
   peeee
   peeee
   peeee
   seeee
 Color: Blue
- Id: 7
 Coordinates: |
   eeeee
   eeeee
   eeeee
   ppeee
   ppeee
 Color: Orange
- Id: 8
 Coordinates: |
   eeeee
   eeeee
   ppsee
   eepee
   eepee
 Color: Purple
- Id: 9
 Coordinates: |
   eeeee
   eeeee
   eeeee
   eppee
   ppeee
 Color: Red
- Id: 10
 Coordinates: |
   eeeee
   eeeee
   peeee
   ppeee
   epeee
```

Color: Red
- **Id**: 11
Coordinates: |
eeeee
eeeee
eeeee
peeee
sppee
Color: Purple
- **Id**: 12
Coordinates: |
eeeee
eeeee
eeeee
ppeee
peeee
Color: Red
- **Id**: 13
Coordinates: |
eeeee
eeeee
pppee
pppee
pppee
Color: Turquoise
- **Id**: 14
Coordinates: |
eeeee
eeeee
eeeee
eeeee
ppppp
Color: Turquoise
- **Id**: 15
Coordinates: |
eeeee
eeeee
pppee
eepee
eepee
Color: Purple
- **Id**: 16
Coordinates: |
eeeee
eeeee
eeeee
epeee
pppee
Color: Purple
- **Id**: 17
Coordinates: |
eeeee
eeeee
eeeee
epeee
pspee
Color: Turquoise
- **Id**: 18
Coordinates: |
eeeee
eeeee
eeeee
ppeee
epeee
Color: Red
- **Id**: 19
Coordinates: |
eeeee
eeeee
epeee
ppeee
peeee
Color: Red
- **Id**: 21
Coordinates: |
eeeee
eeeee
eeeee
ppeee
epsee
Color: Purple
- **Id**: 22
Coordinates: |
eeeee
eeeee
pppee
pspee
pppee
Color: Pink
- **Id**: 23
Coordinates: |
eeeee
eeeee
eeeee
eeeee
ppspp
Color: Pink