**Sea Adventures- DevLog**Lyudmil Pashayanov

# Introduction:

My name is Lyudmil Pashayanov and this Dev Log document is done for Fontys Semester 7 Minor: Game Design and Technologies. The document consists of 2 major topics: Monetization Design for a hyper casual mobile game and SOLID Technical Architecture in the game. The [DOT research methods](#DOT) were used to research all topics needed for the project.

In the first part I focus on the monetization research and design, explaining:

1. how I will present Advertisements to my players in a non-intrusive for the users, but at the same time profitable for me way
2. What currencies and what microtransactions there will be, so that the game could make money
3. How is the game core loop designed, so that it provides opportunities for profit

The second part of the Dev Log focuses on the technical part of the game, where I will share:

1. Why and How I implemented SOLID principles in my game.
2. How I created a simple Enemy AI
3. How I created leveling design easy and fast.

# My Business Strategy Research and Design:

## Overview:

I would also like to create small economy within the game, which would be able to later monetize the game. For example, create a currency so important and valuable so that a player spends money on it or develop the flow in the game in such a way that makes the user want to watch an Advertisement.

My focus on this project will be to use the available products analysis, literature study, Best Good Bad Practices methods to research and design a strategy for monetizing a game. There are several wide-known business models for achieving that goal ([Appendix 1: Business strategies for monetization for your game](#appendix1)).

The expert interview method was also applied for this topic of the project (monetization). After the initial design of monetization strategy, the project was handed in to an expert economist in the gaming industry currently working in Gamehouse. It was done with the view to see what he thinks about the prepared monetization design for my project.

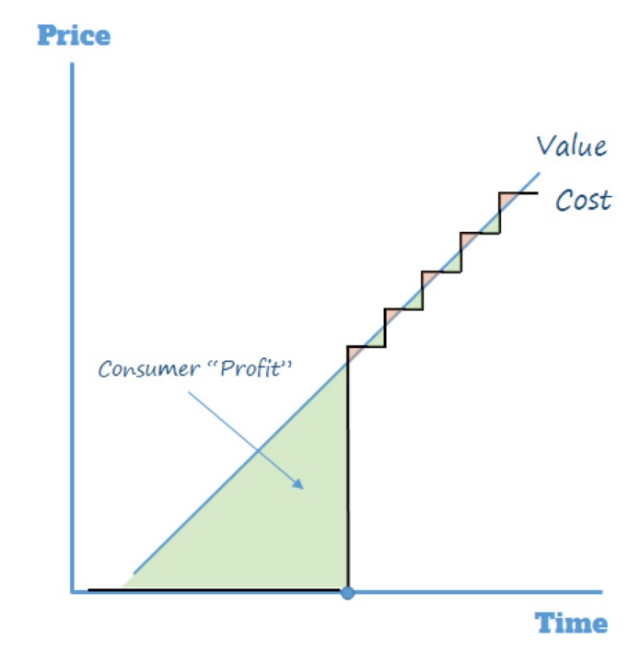
The overall feedback was positive, with remarks about what I should be careful about. In the following lines, I will share what was changed after the expert interview and what was approved from the expert.

## My strategy:

The business strategy I will be designing to monetize the game, will work as making the game **free** **to Play (F2P)** and relying on **Micro transactions and Rewarded Video Ads**. The decision to make Sea Adventures **F2P** comes from several factors:

* I want players to set their own price for the game: Those who want to pay $1 will pay that, and those who want to pay $1000 will pay that.
* I hope that I have a large userbase, when the game is free, and by that the players create in game value just by being there, even if they don’t pay.
* I believe that I can generate interest in players to spend maybe thousands of dollars, which will offset the fact that most people will never pay.

After analyzing the available games, doing a library research, it seems that this kind of hybrid strategy works very well for mobile hyper casual games([research source](#ResearchStatsOnMonetization)), and it can be seen applied in a lot of successful mobile titles as well ([Wordscapes](https://play.google.com/store/apps/details?id=com.peoplefun.wordcross&hl=en), [Angry Birds](https://play.google.com/store/apps/details?id=com.rovio.angrybirdsspace.ads&hl=en), [Piano Tiles](https://play.google.com/store/apps/details?id=game.piano.black.tiles.music&hl=en)). Also receiving approval from an expert economist from Gamehouse I am positive that the design is on the right path.



“I get an abundance of consumer profit in the beginning with a free-to-play game because the cost to me is zero. However, as I progress, I get pinched and I end up paying. At an aggregate level (i.e. across all players) users are paying for the product after trying it for free. Users get all consumer profit up front, but the profit is then split between consumer and producer later”

### Currencies:

**Upgrade points-** Granted at the beginning of a level. Used to create and upgrade defenses and upgrade your ship during the level. All upgrades done during the level are not persistent and are lost when the level has finished. Upon a finished wave, more **upgrade points** are granted, for upgrades to be done in that level. Upon winning a level, the rest over upgrade points are granted and converted to **Coins**.

* Idea behind this is to make the levels harder (play smart, plan out moves) by forcing the player to spend fewer upgrade points during the level, so that she/he can receive more **Coins** at the end of it. This mechanic will bring more value to the game as it will force players to player riskier.

**Coins**- Soft currency, but harder than **Upgrade Points**. you receive from the rest over **Upgrade points** you have when you finish a level. This currency is planned to be used the most, therefore the design is driven to make people buy **Diamonds** and then buy **Coins** withthe **Diamonds.**

* With **coins** you can buy ***new abilities***, *which are vital to game progression. Coins should be perceived as the most valuable currency in the game. This is why the game should be balanced in a way that no matter how much* ***upgrades points*** *you spend during a level, the abilities you bought from the shop for coins should be perceived better.*
* *You can also buy* ***Keys*** *to enter a level****,*** *as well as**customize your ship.*

**Keys**- The currency you pay 1x for starting a level. There will be max amount of this currency. It will recharge *overtime or c*ould be bought for **coins**. This system is often found in hyper casual mobile games and has proven its value. This system was also approved by the expert economist:

* Idea behind is to prompt a **Rewarding Ad** if the user has used all his **Keys** during his play session, to receive **coins** and then use them for a **Key.**
* If people want to play more, then they need to pay money for **coins** and buy more **Keys**.

**Diamonds-** Hard currency you can get by paying real money. Used for buying more **Coins.**

##### Feedback from the Expert Interview:

* *It's a good practice to have this Real Money --> Diamonds --> Coins --> Keys, Upgrades system. Having more distance between money and the currency players use more often is a good way of having players spend more, having more opportunities for rewards of different value and overall increasing the speed of currency flow.*You are building a hyper casual type game with mechanics borrowed from the Tower Defense genre. With this in mind, I think your strategy fulfills the basics: It has currencies that seem like they can hold heir value and not cannibalize each other.

*Tip* from the Expert Interview to keep in mind:

* It's extremely important that the design allows for upgrades that can be purchased both with coins and with upgrade points that do not cannibalize each other.
  + Are the upgrades only valid in that level or are they persistent? If they are persistent, they are purchased with coins. Make sure none of the temporary upgrades does anything that is similar to any persistent upgrade.
  + Are upgrades direct (e.g., a weapon) or indirect (e.g. a damage bonus to a weapon)? Typically, indirect upgrades are more related to metagame and thus make more sense to have purchased by coins.

## Core level Loop Of “Sea Adventures”:

### Start of a level:

Player loses 1x **Key** in order to start the level.  
The player starts with pre-determined amount of **upgrading points.**

### Preparation phase:

Player will have to prepare for the big waves of enemies coming to attack. The preparation consists of:

1. Upgrading your ship abilities/stats: By spending **upgrading points**.
2. Putting defenses on the map: By spending **upgrading points**.
3. Upgrading your already active defenses: By spending **upgrading points**.

### Wave phase:

During the wave, the player actively attacks incoming enemies and defends the main point of interest (island). While destroying enemies, the player receives **upgrading points**, which could be spend on the next preparation phase.

### Lose State:

In case the player fails to complete the level (dies or the island dies), he is prompted to watch and **Ad** and receive Gacha rewards (extra life/double **coins**/25% more **coins**/ 10% more **coins**, etc.). If he refuses, all of his **Upgrade points** are lost and the user is sent to the main menu.

### Win State:

In case the player successfully beats the level, all his left-over **upgrading points** are converted to **coins**.

## User Scenario:

In the user scenario below, you can see how the currencies are going to be rewarded and how the player will spend them.

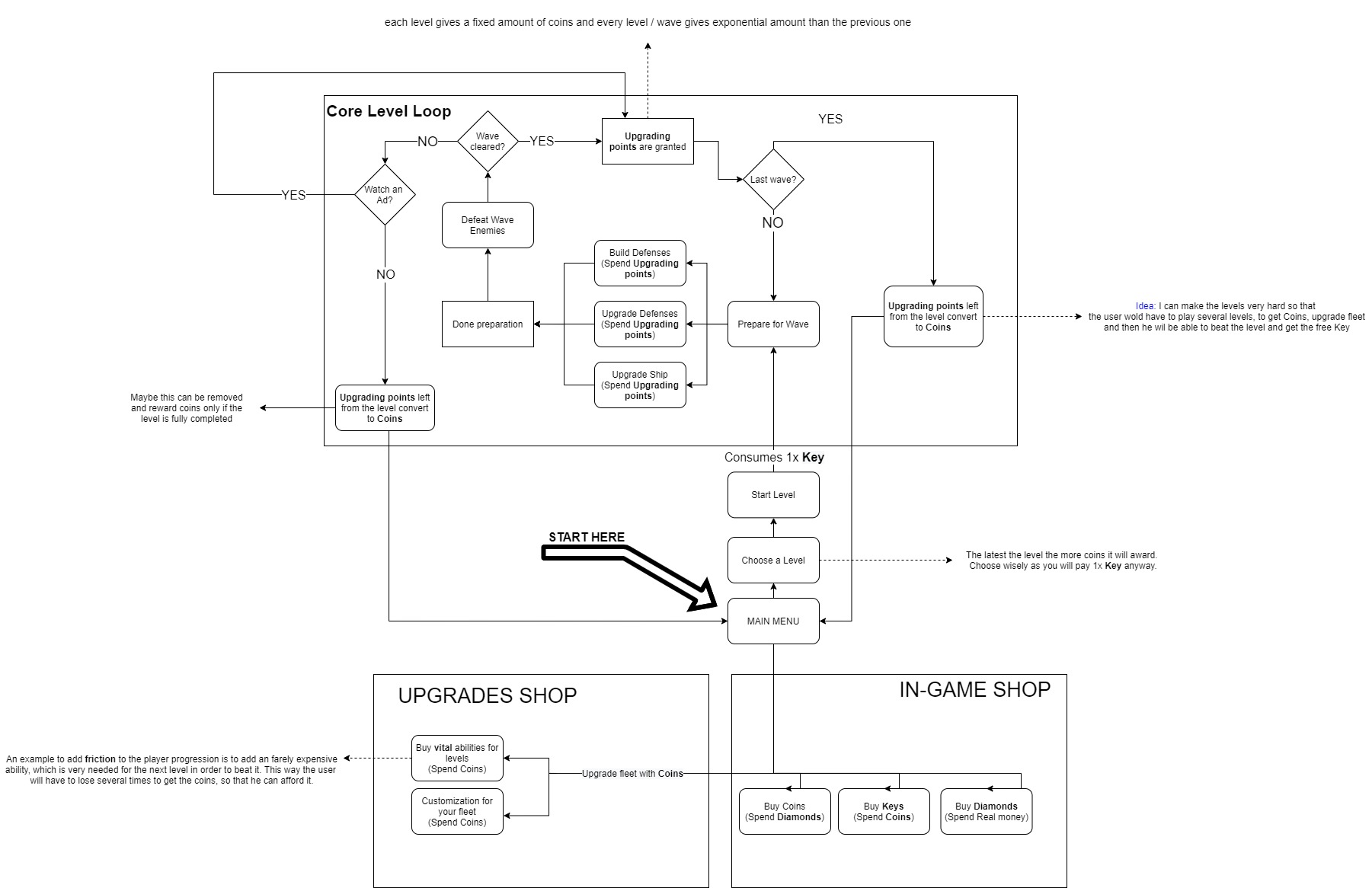


Figure 1: User scenario based on notes focused on monetization

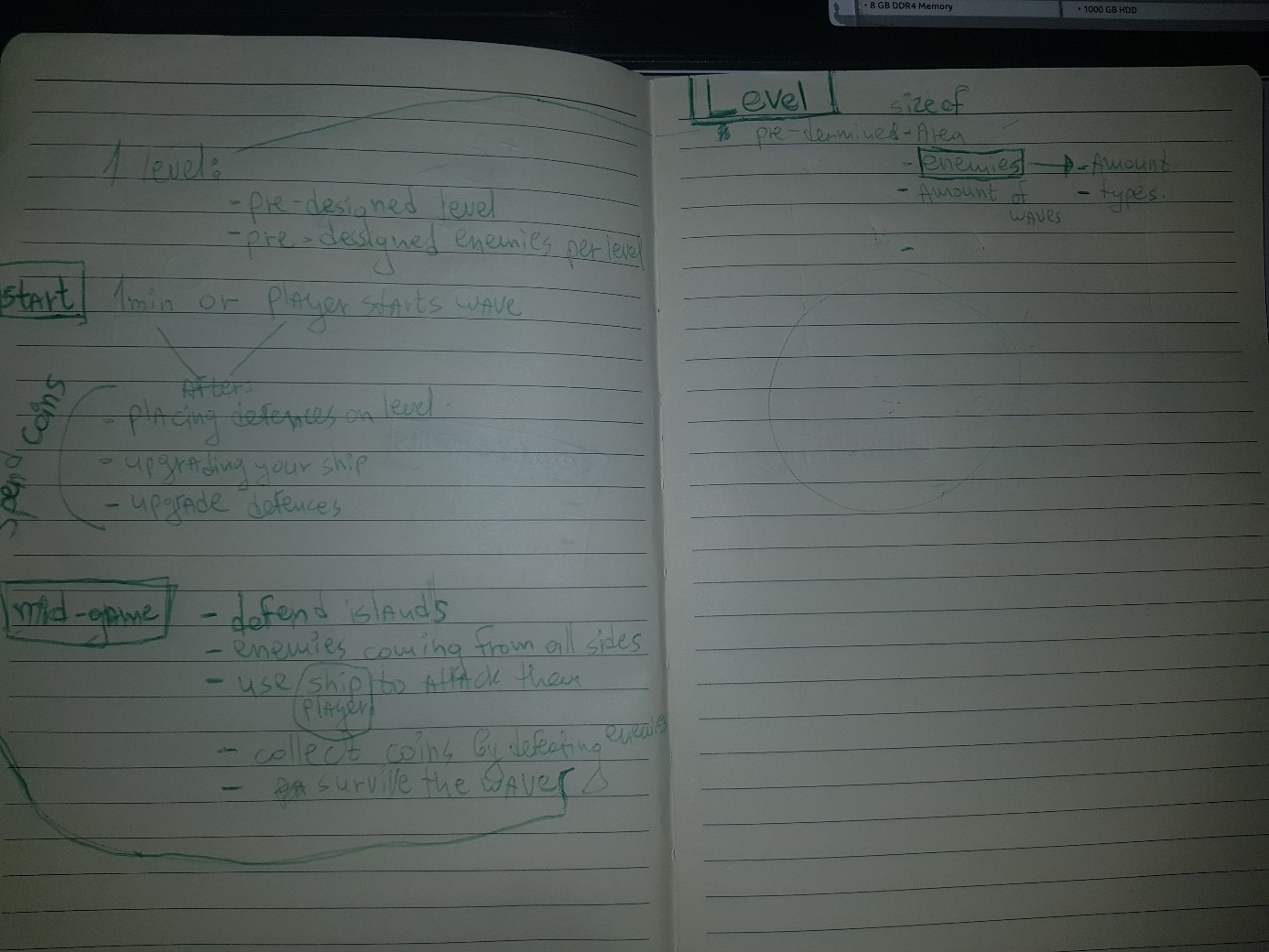


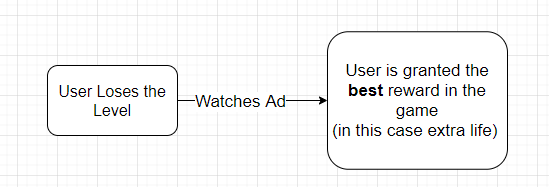
Figure 2- Core Loop notes

### Ads:

By using the ‘Best good and bad practices’ method when it comes to Ads in games, I reached the following conclusions: I do not want to ***force*** the user into watching an Ad, as eventually this will make him/her quit my game. On the contrary, I want to present the Ads in the game in such a way so that my users want to watch them -> I will give them a very valuable reward after the Ad and give them the option to cancel the Ad and not receive the reward.

#### Rewarded Ads in Sea Adventures:

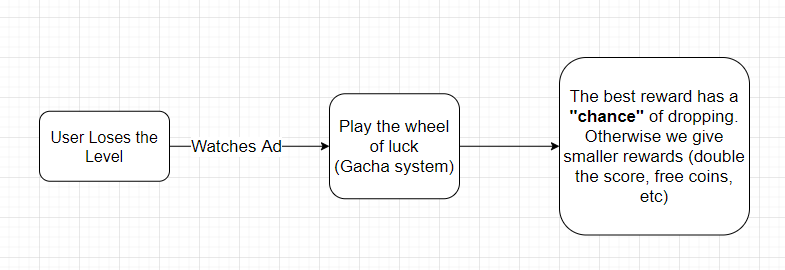
In Sea Adventures after the user dies during the level, he will be given the choice to continue, as if he never has died, in exchange of watching an Advertisement ([user scenario](#userScenario)). I see in lots of games ([Mario Cart](https://mariokarttour.com/en-US), [Jetpack Joyride](https://halfbrick.com/our-games/jetpack-joyride/)) extra life as a reward is valued enough to prompt the user to watch an Ad and from personal experience makes him feel rewarded



#### After Applied feedback from the expert interview:

Feedback from my interview was the following: - *The continue system for ads is a good idea, but you need to consider the repercussions of this system on the overall design and impact on monetization: How many continue options do players have? Will this make it less likely they spend coins/upgrade points on upgrades? Will it affect their win rate and thus force you to create more content more quickly?*

In order to keep the ads having a bad impact on my in-app purchases, I decided to control when the user will get an extra life and when not. To do this I used a Gacha system. That is a great way I can control what % of the Ads will grant the extra life and when it will grant a smaller reward which wouldn’t affect my monetization. However, this system is notorious for making players dislike the game, as they basically gamble when watching the Ad and don’t know what they will get in return. In order to avoid player churning, I would have to play test and balance the “wheel of luck” so it doesn’t make the players feel cheated.



## Features Introduction:

As it is very important that your items are perceived as more valuable as possible, it is very important to introduce them in the right way.

|  |  |  |
| --- | --- | --- |
| **Level** | **Introduced feature** | **Goal** |
| **1-3** | * Basic Attacks * Basic Enemies * Movement controls | Tutorial |
| **4-7** | * Upgrading Points * Special Attack | Tutorial |
| **8-10** | * Special enemy | Shows that new enemies appear |
| **Main Menu after 10th level** | * Upgrades shop * Buy the ‘’traps ability’’ from the upgrades shop | Tutorial  + Emphasizes where you can spend your coins  + By unlocking it, it makes players curious.  + It functions as a reward |
| **11-13** | * Basic Trap | Tutorial to what you have bought from the ‘’upgrade store’’ |
| **14-20** | * New special Trap * New enemy (killable **faster** withspecial trap) | Emphasizes that you need to upgrade and use traps to win levels |
| **20-…** | … | … |

From the expert interview a positive note was made about the introduction of the shop and overall features onboarding.

# Game creation:

## Overview:

I am inspired to create a **Tower Defense**, **Strategy** game. Main idea is that you have your ship (*player*) in open waters (*level map*) and you have an island (main point of interest), somewhere on the level map, which must be ***defended***. Just like in “**Orcs Must Die**” the player, would be able to place ***defenses*** on the map, which will help him when the pirates (*enemies*) come. I would like to be able to introduce several types of ships, with which the player will be able to play with (*attacker, defender, carrier*), all of which will have different advantages and disadvantages. The level will be similar to a **Plants vs. Zombies** level, where there are several waves and the player needs to survive until the end.

## Enemies design:

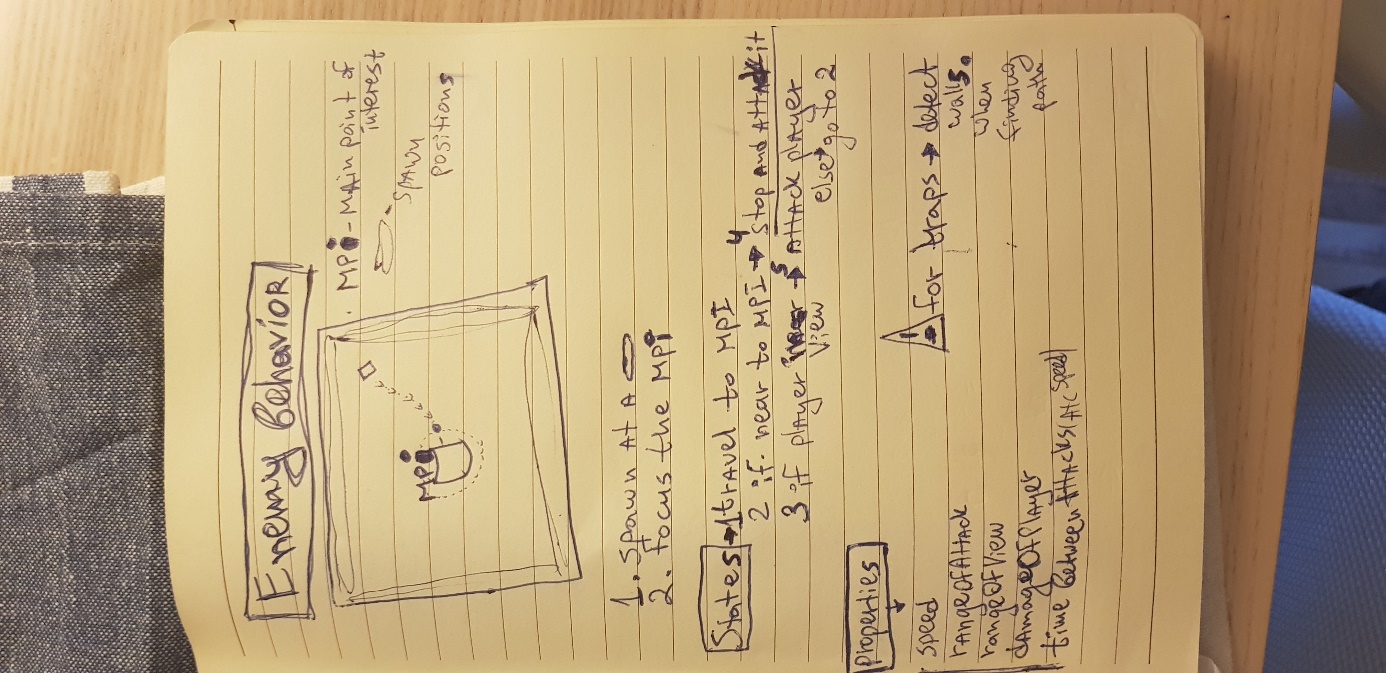


Figure 2: IT Architecture sketching method (Workshop) on Enemies in Sea adventures

### Enemies AI:

### Pathfinding:

My enemies should spawn on the edges of the level and go to the center of it and attack the Island. As I want to have traps in my game, I had to create my enemies go to the island, but in the same time go around the traps (walls) on the map. In order to do that I had to create some kind of a pathfinding system. As I found out it will be a lot of work to create, I found an already made one by Unity ([NavMesh by Unity](#brackeys)).

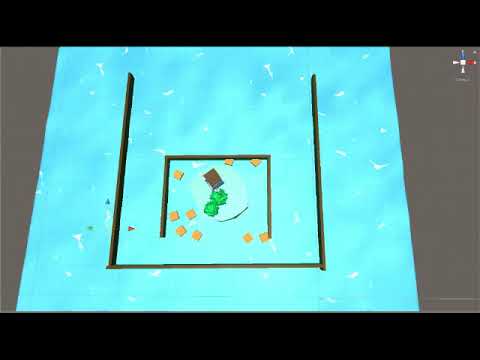
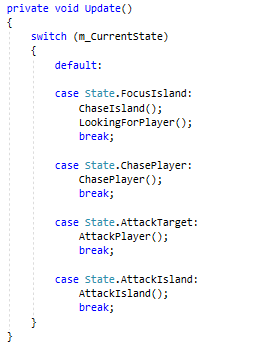
[](https://www.youtube.com/embed/JTHAcrRyOL8?feature=oembed)

Figure 3: My enemy pathfinding functionality

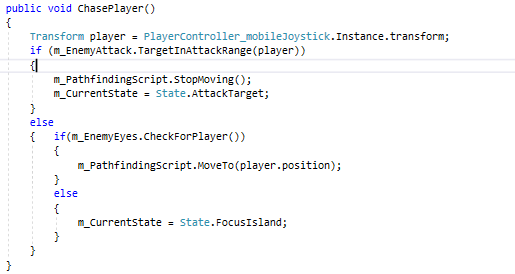
### Enemy States:

My enemy AI states are designed to be the following:

1. Travel to the island.
2. If you see the player, chase him.
3. If you get near the player attack him.
4. If you don’t see the player go to 1.
5. If you reach the Island, Attack the island until you defeat it.

As a for now it is planned that all the enemies will have the same AI behavior, I wanted to choose the **fastest approach** to simulate enemy behavior and the simplest way to create that turned out to be an AI state machine([Simple Enemy AI](#codemonkey), [AI State Machine](#JasonWeimann)):  


Where the [states](#AIstates) of the AI change from similar to this function:



In case I want to create enemies with another Behavior/AI (other states), then I would have to change this approach to **Behavior tree approach.**

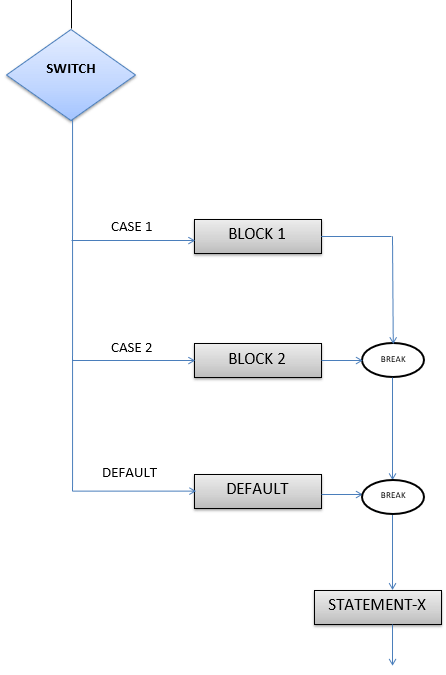


Figure 4: My simple represented by a graph.

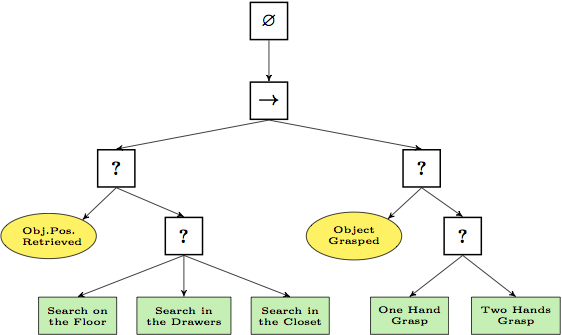


Figure 5: Behavior tree AI represented by a graph

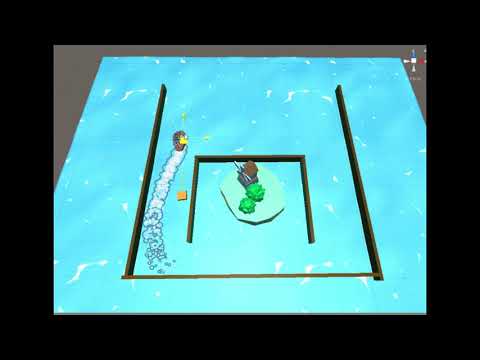
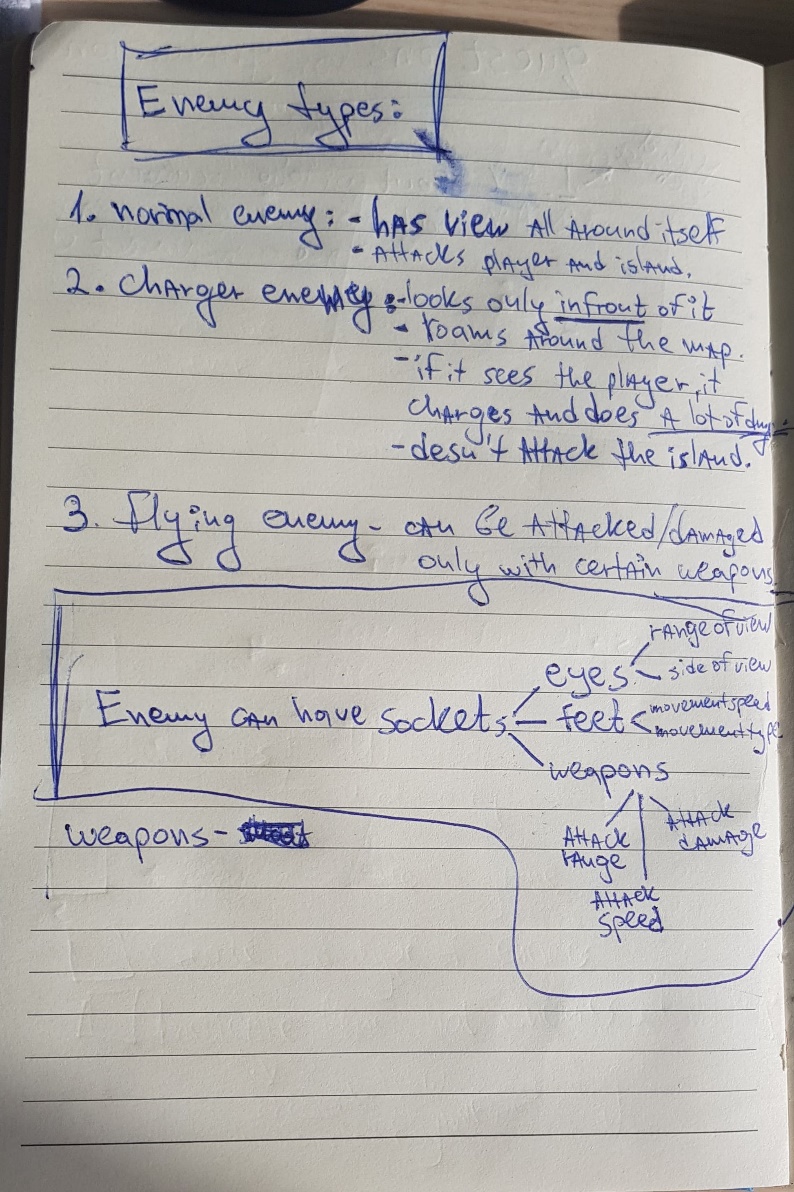
[](https://www.youtube.com/embed/aIzvKUoVwvI?feature=oembed)

Figure 6:   
Red line = not in range -> going to the island.  
Yellow line = player in range, but not in view -> go to the island.   
Green line = player in range and in view -> attack player.

# Code Talks:

## Enemies Design:



## SOLID Code:

By design I want to have different types of enemies. If I want to scale the game and follow good code practices, I want my enemies to be easily extendable and easily modifiable. In order to do that I did a Design patterns research on SOLID principles.

### Open Closed Principle:

I implemented the [O**pen Closed Principle** **(OCP)**](#OpenClosedPrinciple)for the development of the enemies, because it allows me to not change my AI class (closed for modification), but at the same time, extend and modify the behavior of my different types of enemies very easy (open for extension).

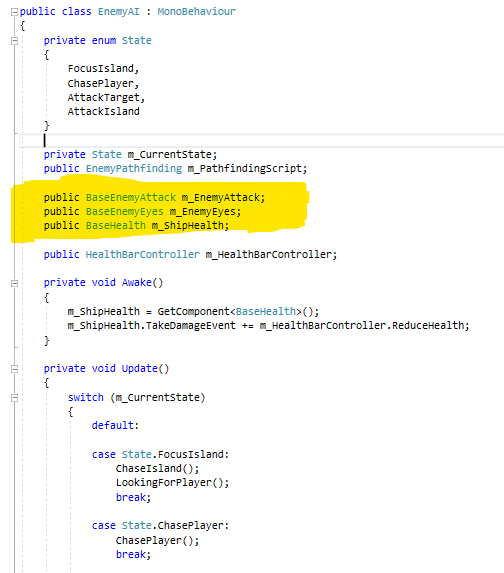


Figure 4: Enemy AI operates with only abstract classes, so that I don’t have to modify it when a new enemy is introduced.

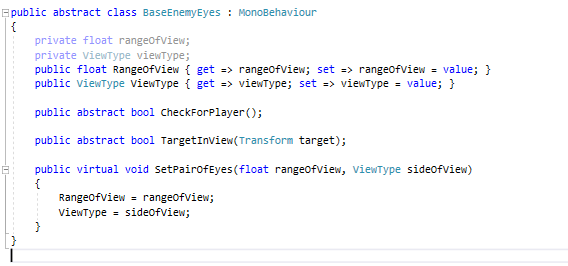


Figure 7: Abstract class implemented by every new type of enemy, which wants to use the AI

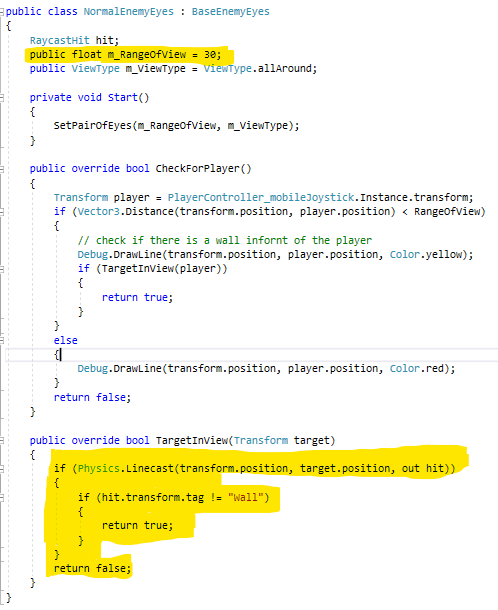


Figure 8: One enemy type could have its vision blocked by walls, so then it won't see the target and have a great view range

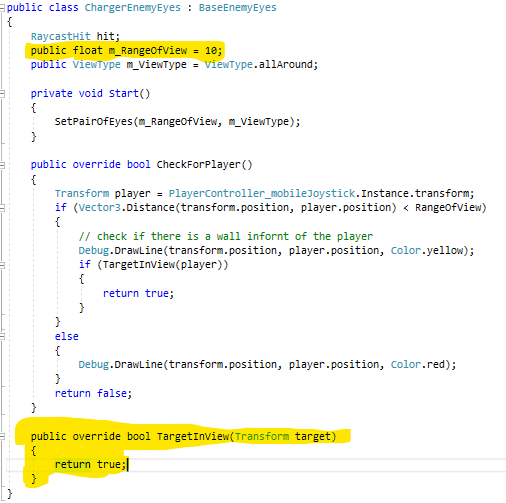


Figure 9: Another type could be that it has smaller vision, but it can see you and attack you through walls

In the latter code examples, both enemies will work correctly without modifying the Enemy AI, as it only works with the abstract classes. This approach allows me to fast create enemies without constantly modifying my enemy AI class:

### Single Responsibility Principle:

As you see in the latter examples, my classes are responsible only for one very specific behavior. A class for the enemy eyes, enemy attacks, game object’s health, etc. From a Design Pattern research ([Single Responsibility principle)](#SingleREsponsibility) and seeing its advantages in my game, I reached the conclusion that this approach has the following benefits:

* **Readability** – Classes are easy to keep between 20-100 lines when they correctly follow SRP.
* **Extensibility** – Small classes are easy to inherit from, modify, or replace.
* **Re-usability** – If your class does one thing and does that thing well, it can do that thing for other parts of your game.

### Observer pattern:

This is a very popular pattern, which I learned by doing a Design Pattern research, and decided I want to learn and implement it.

I my case bellow I have implemented it in my ‘’BaseHealth’’ class so that I every time when the object with the ‘’BaseHealth’’ class attached (player or enemy) is taking damage, all classes which are subscribed to the ‘’TakeDamageEvent’’ are notified and act respectively: reduce the HP Bar, play animation for taking damage, play sound for taking damage. All of that is done so that it obeys the single responsibility question and as well keep the code easy to read.

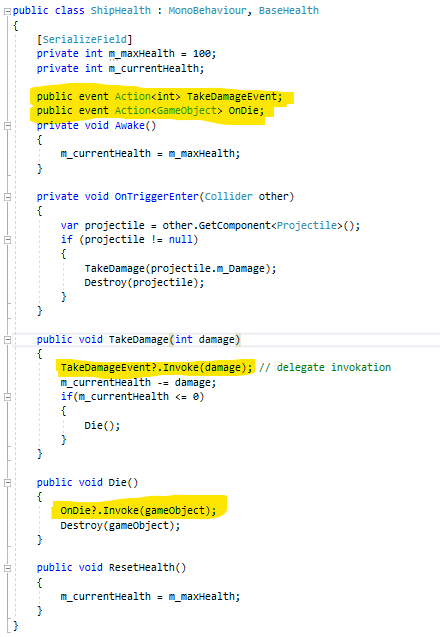


Figure 10: Notifying of all the observers of those events

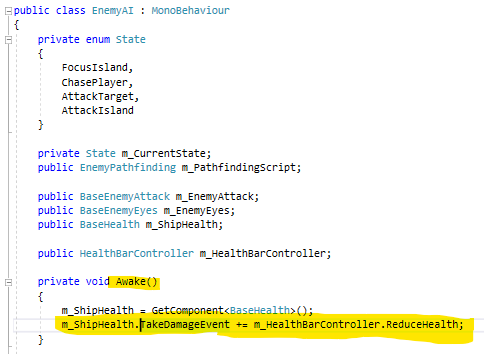


Figure 11: On creation of the enemy, we subscribe to the event, so that when it triggers, we visually reduce the HP bar

## Levels in code:

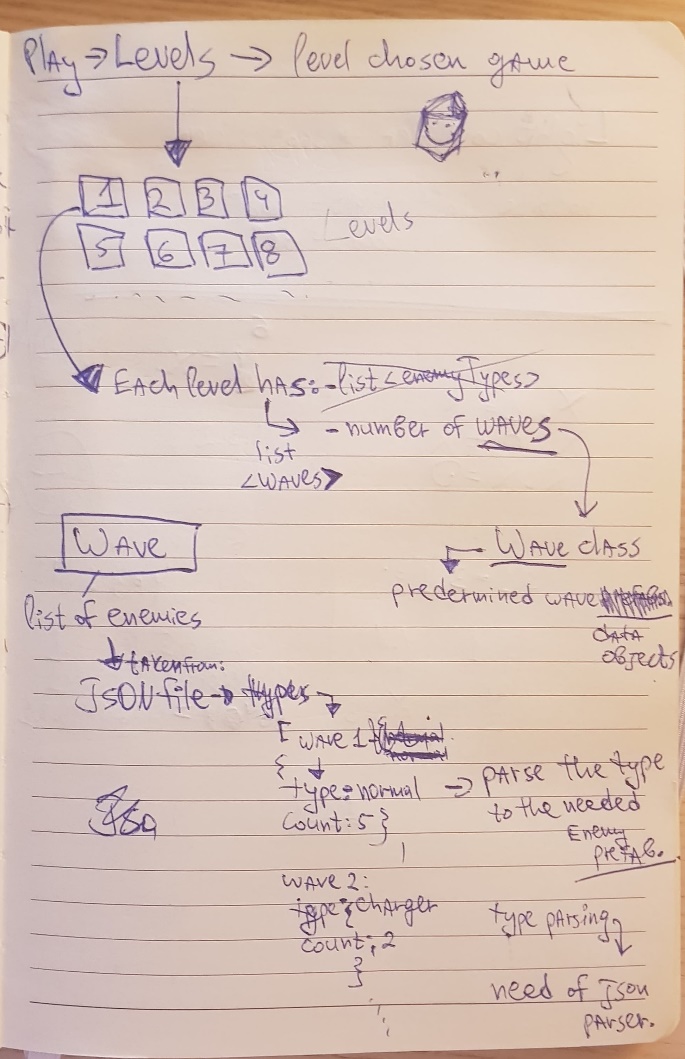


Figure 12: IT Architecture sketching method (Workshop). Design on how I want to start a level + What a level data structure will be.

From my personal experience, working at Blue Giraffe, as a Game developer, I know that it is crucial to any game to be flexibly and be able to easily modify its levels, so that when you spot on which level people die a lot or beat the level easily, you can change the levels difficulty to suit your needs.

By design, I want to have full control over my levels, so that in the future, when I had multiple rounds of playtesting, I can balance the levels effortlessly.

That is why developed the levels to be read and started from a JSON File ([Levels Design](#figure10)). JSON is an easy to read and learn format known to almost everyone in the gaming industry. Having my levels on a JSON allows a person without technical knowledge to have flexibility and change the levels difficulty, just by changing the JSON values.



Figure 13:Level data in a JSON File



Figure 14: Classes Level and Wave creation from the JSON file

# Reference:

1. Business Models, Author: Anuj Nawal (date: 21.06.2018) Ref: <https://www.feedough.com/how-to-choose-the-right-business-model-for-your-game/#:~:text=The%20six%20types%20of%20game,gotta%20pay%20for%20the%20oars>
2. Revenue information of mobile apps, Author: Sensortower Ref: <https://sensortower.com/>
3. Top Publishers Rely on Video Ads for Mobile Monetization, Author: ExchangeWire PressBox (date: 12.02.2020) Ref: <https://www.exchangewire.com/blog/2020/02/12/top-publishers-rely-on-video-ads-for-mobile-monetisation/>
4. Designing For Vs Designing AROUND Player Behavior, Author: Game Dev Underground (Date: 12.06.2020), Ref: <https://www.youtube.com/watch?v=0Ie0jQmI8fA&app=desktop>
5. Star Wars Battlefront II: The Only Analysis You Need to Read, Author: Michail Katkoff (date 21.11.2018) Ref: <https://www.deconstructoroffun.com/blog/2017/11/20/deconstructing-the-star-wars-battlefront-ii-controversy>
6. Unity NavMesh Tutorial – Basics, Author: Brackeys, Ref: <https://www.youtube.com/watch?v=CHV1ymlw-P8>
7. Simple Enemy AI, Author: Code Monkey, Ref: <https://www.youtube.com/watch?v=db0KWYaWfeM>
8. AI with state machine, Author: Jason Weimann Ref: <https://www.youtube.com/watch?v=YdERlPfwUb0&t=209s>
9. Unity3D - SOLID Code - Open Closed Principle, Author: Jason Weimann Ref: <https://www.youtube.com/watch?v=wYkzeKghjsI&list=PLB5_EOMkLx_WjcjrsGUXq9wpTib3NCuqg&index=2>
10. Observer Pattern - Game Programming Patterns in Unity & C# , Author: Jason Weinmann Ref: <https://www.youtube.com/watch?v=Yy7Dt2usGy0&t=612s>
11. Single Responsibility principle, Author: Jason Weimann Ref: <https://unity3d.college/2017/01/10/unity3d-architecture-srp/#:~:text=The%20single%20responsibility%20principle%20states,narrowly%20aligned%20with%20that%20responsibility>.
12. DOT research methods, Ref: <http://ictresearchmethods.nl/Methods>

# Notes (pictures):

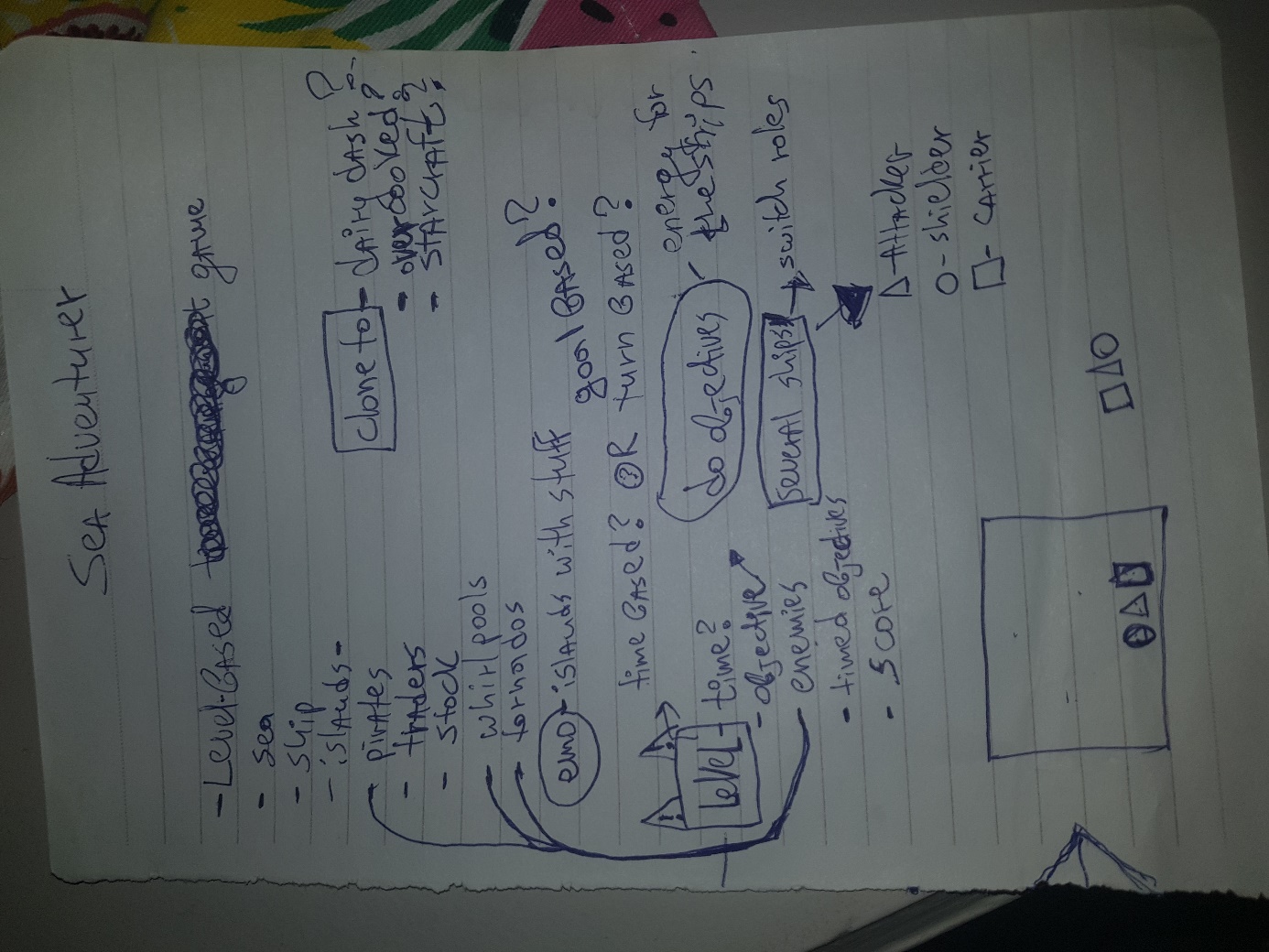


Figure 15-The very first Brainstormed ideas for Sea Adventures

# Appendix 1- Research on business strategies for monetization within the game industry:

## Overview:

My focus on this project will be to research and design a strategy for monetizing a game. There are several wide-known business models ([business models](#BusinessModels)) for achieving that goal:

**-Microtransaction** is a business model, which represents the following: The game has *no* *cost* to download, but by selling valuable virtual goods, the game studio earns money.

**-Showing Ads** within your game.

-Make the game **paid (premium game)**.

## Micro transactions Strategy:

Successful games (revenue-wise) such as the Supercell games, ‘Fortnite’, ‘Overwatch’, all use Microtransactions to make money ([Sensortower](#Sensortower)).

Usually in games with microtransactions there are 2 types of currencies- Soft and hard currency.

* Soft currency focuses on building the **basic structures** for play and gives progression to the non-payers (people who do not buy the hard currency) with a slower grind.
* Hard currency aims to maintain value by remaining a rarer resource, which can give a better experience to the player, by giving access to exclusive items, gacha, skins and/or focuses on time/grind **reduction** within the game.

However, it is very important to maintain balance between the payer and non-payers. Non-payers should not feel that if they do not buy exclusive items and features, they are less strong/good at the game, in comparison to the people who are paying in the game. If this happens, non-payers are more likely to leave the game and never come back playing it as it feels to be “Pay to win” game.

By separating the hard- from the soft-currency, the designer can be sure that the payable currency (hard currency) is not over-inflated through actions in game and can be controlled with more ease.

However, some games allow conversion between currencies. This is risky as both currencies need proven balance in advance. Otherwise you could destroy value of your premium currency or over-inflate your soft currency, but it can move economy focus on retention (non-payers).

## Ads Strategy:

### Overview:

By showing advertisements to your players, you stop the fun in your game, and this is perceived as an irritating aspect in your game. Therefore, Ads should be used smart by the developer.

### Ad Strategies:

A strategy to make a lot of money out of your game with Ads, is if you ***forcefully*** show the user an Ad every once in a while, (after each 3rd level for example). Although, it seems that this way of the user will see a lot of Ads, this strategy will most possibly work short-term only. A lot of users do not tolerate this spam, no matter how fun the game is and later on they quit the game. However, if the game has a constant flow of new users and the company does not care about how it will be perceived by the public, then this strategy could be the most profitable one.

However, there is another way to integrate Ads within your game. If the goal of the developers is to keep their players playing their game, the Ads should be presented in a delicate way. The user should see what *valuable reward* is waiting for him *after* the Ad and have the *option* to refuse or accept watching the Ad. This strategy is called ‘‘**Rewarded Video Ads’’** and it gives the user the feeling of control over the game and if he runs the Ad it will be because he wanted to, not because the game forced him. Here comes the part where the company should integrate the Ad in such a way that the user would *always (as often as possible) want to watch it.*

## Premium Game Strategy:

### Overview:

If the users want to play your game, they must pay for it first.

### Examples of who is using this strategy:

You need to have a very good advertisement, in order to make the people buy your game, before they can play it. Usually this marketing strategy (Premium games) is picked by triple A studios, which already have many fans, which will buy their game. Those companies also spend a lot money in marketing campaigns to make a fuzz and hype the people for the game they are releasing. Such companies are for example: “EA games”, “Ubisoft”, “Blizzard”, etc.

However, if you are an infamous studio, but your game is very unique and offers something which hasn’t been seen yet, with good advertisement you can get people to buy your product. An example for such games is: “Cuphead”, being unique with their visuals, “Fall guys”, being unique with their multiplayer competitive levels, “Dreams”, being unique with the creativity they give to their user and much more.

### Goal of the strategy:

Using this strategy, the goal of the seller is to give a good run for the player’s money. For example, Player pays 10-60 dollars up front for a game. The value they get out of it is measured in the subsequent time they get out of it.

*“Let’s take a theoretical example of one of these types of games - let’s say I buy the latest Zelda game, and I pay $60. Additionally, let’s say I value 1 hour of entertainment at $1. If I get 60 hours’ worth of entertainment, I “break even”; meaning that I feel like I got my money’s worth. (see graph below) Until then, I’m experiencing a consumer “loss” - but as soon as I reach the break-even point, any additional amount of time that I play that game I experience a consumer “profit”. I’m getting more than I bargained for, which is the ideal state. The gold standard games that we all love would break this barrier and provide us with an abundance of profit.” ([Deconstructor of fun](#premiumGameGraph))*

