A Component-based 2D Game with Data-structures, AI and algorithms

Ammar Muhsin
Armin Alicajic
Joakim Schack Betzer
Jonas Dige
Mohamed Macow
Mohamed-Yonus Rafik
Peter Ratgen

March 2021

Gruppenr: 6
Semester: 4
Kursuskode: T510034101
Fakultet: Det Tekniske Fakultet
Universitet: Syddansk Universitet
Projektperiode:
Institut: Mærsk Mc-Kinney Møller Instituttet
Uddannelse: Software Engineering

Contents

1 Abstract

2 Introduction

The objective of this project is to create a 2D game utilizing a component-based architecture. The game the group is looking to implement a top-down RPG shooter, themed around the coronavirus pandemic.

2.1 Related Work

To find which elements the solution should contain, related work is examined. In picking related work, we pick games that are 2D-based and has some relation to RPG or shooter-style games.

2.1.1 Gameplay

Hotline Miami Hotline Miami is a top-down shooting game. In this title, the player must clear rooms of enemy entities to complete the levels.[1] The player can freely move around the map, where entry to multiple rooms is possible. To help the player defeat the enemies, various weapon entities spawn around the map, and it is possible for the player to spawn with a weapon when starting the level. Different weapons have varying attributes, that depend on the type of weapon. Therefore different situations will require an arsenal of weapons, which the developers have fixed by implementing a way to throw weapons to grab others.

As the concept of the game is fairly simple and supplemented by a story line, the player engagement must be maintained by other means. Hence why a score system has been made, to appeal to players who compete by having the highest score possible. The score is a variable, which increases when the player kills a hostile entity. The score [2] increases depends on a few variables, a few ahttps://www.overleaf.com/project/603f6f0733b06374fa5b11camong them being:

- Reduced time between kills in a combo
- Using various weapons to complete the level
- Being exposed to other enemy entities when killing

To make the game more enjoyable, the enemy entities have implemented different AI behaviors. Enemies have varying temperaments and fighting styles, which requires the player to handle each situation in an appropriate matter. The most important AI is arguably the path finding, which allows the enemies to move around the map. If the player should be discovered, enemies will flock towards the player and try to kill him.

Binding of Isaac The Binding of Isaac is a rogue-like RPG game. [3] The story revolves around a young boy named Isaac and his very christian mother. One day the mother hears a voice from above, which er that to prove her devotion, she must sacrifice Isaac. Isaac witnesses this conversation, and just as his mom enters his room, he jumps into the basement to escape. This is were the game play starts, and the player gains the ability to control Isaac. The goal of the game is to fight the different monsters throughout the levels of the basement. As the player progresses further into the game, new characters, power-ups and boss-fights are unlocked. At the start of a run the player is able to choose from a set of characters, each with their own strengths and weaknesses. The map is then loaded with random generated rooms that the player has to fight his way through. The monsters are randomly generated within the room, and each monster has their own unique AI behavior. Some monsters use simple path walking, while others are more advanced and implements different kinds of trajectory prediction. The Binding

of Isaac gained its popularity by being a very enjoyable and re-playable game. By having a lot of power-ups, which interacts with each other, the game ensures that each run feels nothing like the old ones. The difficulty also increases as the player progresses further into the game, challenging the player to see how far they can go.

2.1.2 Top-down genre features

Every game has its own course of actions, in which a player has to undergo. Within their adventure, they will meet several challenges and personal development opportunities. Games like these develop gradually in challenge, the longer a player progresses.

What characterizes top-down based games is the fact that there is a single player whose purpose is to get through several levels and challenges. This is done using weapons and possibly other entities that help the player on the right path and make it possible to reach the desired goal. Top down RPGs are only seen from above, so first person based interactions that are otherwise known from newer games, do not exist.

Various weapons and entities, such as "enemies" and "objectives" are also important factors in top-down RPGs, which is why there is an increased focus on the development of these entities early in the phase. The enemy is usually the entertaining and the challenging aspect of the game, which follows the overall development in the game, and secures dynamics in the story. The further the player reaches in the game, the more does the enemies develop and require greater resistance from the player. The main concept of the enemies, is to prevent the player from developing and achieving the overall goal.

Every course of action has a consequence, as to why several outputs from decisions has to be defined. Those outputs are then a base for several other outcomes that can affect the general outcome of the game. Such as the choice of which missions to execute, which bosses to confront and if possible, which story line to chase. Every choice the player decides to do, has some effect on the general outcome. Top-down shooters has its own characteristics, as to having certain goals to achieve by fending off enemies for rewards and personal improvements, in comparison to top-down open-world games where there is no certain objectives that has to be executed in a sequence of time, but a pool of choices and independent actions that athat define a successful game in the genre. player can take to decide its own future.

2.2 Differences

After looking at *Hotline Miami* and *The Binding of Isaac*, and the generalities of the top-down genre we can discuss the differences, between these and our proposal for a solution. The group's proposal is for a game themed around the COVID-19 pandemic. Due to the current circumstances in society, this makes the game very relevant as a tool for information and fun. By creating weapons that mimic real ways to fight the virus, players will become familiar with the tools that actually work. Much like *Hotline Miami* and *The Binding of Isaac* the game will utilize weapons of different sorts to make the combat system work. The solution will not have a way of progression, and will be similar to a sand-box format, where the player can move around the map and fight enemies.

2.3 Solution Domain

A definition of the solution domain is important, because it clarifies where the boundary of our solution lies.

Restrictions have been set that constrain the domain of the solution. These are outlined in the description of the project.[1] These include:

- The game must be a 2D game.
- The game has to include `Player`, `Enemy`, `Weapon`, `GameEngine` and `Map` components.
- The components `Player`, `Enemy` and `Weapon` must implement a provided interfaces, to allow update and removal during run-time.
- The games must utilize a component framework which allows multiple class-loaders and the ability to version components. This gives the possibility of using either OSGi- or Netbeans Module System framework.
- One component should also implement artificial intelligence.

Our solution must conform to these restrictions. However since the game to be created is a 2D-game and must also therefore conform to or incorporate some of the characteristics that define a successful game in the genre. A game in the genre has certain characteristics these are defined in section 2.1.2.

2.4 Domain Model

This task can be done through compiling a domain model diagram.

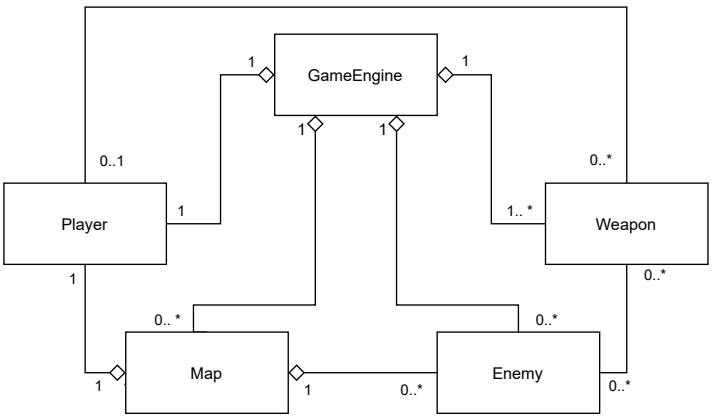


Figure 1: Use case diagram

The domain model diagram helps us understand the real world context the game will be working in. That provides us with a helpful way of communicating our desired product. It is therefore primarily based on our functional requirements, which are further defined in use-cases. Including non-functional requirements is also another factor, when it comes to choosing architecture.

[1] https://drive.google.com/file/d/1NLN048qmOD1hj-u1hL2yCqDQJumOPlEC/view

3 Requirements

The following requirements have been derived from 2.1 Related Work, and the project requirements given.

3.1 Functional Requirements

ID	Description
F-1	The game must implement different entities
F-1-1	The game must have a Player entity
F-1-1-1	The player can be moved around the map
F-1-1-2	The player can pick up weapons from the map.
F-1-1-3	The player can drop the equipped weapon onto the map.
F-1-1-4	The player can hold several weapons at once.
F-1-1-5	The player can damage enemies' healthpoints.
F-1-1-6	The player can interact with NPCs
F-1-2	The game must have an Enemy entity
F-1-2-1	The enemy can use the weapon entity
F-1-2-2	The enemy can damage the player's healthpoints.
F-1-2-3	The enemy must implement the A-star algorithm for path-finding
F-1-3	The game must have a Weapon entity
F-1-3-1	The weapon must be able to damage entities' healthpoints based on the related properties.
F-1-3-2	Weapons should be controlled through the entity in which they are in the possession of.
F-1-3-3	Weapons should have a finite number of ammunition to shoot
F-1-4	The game could have a NPC entity
F-1-5	The game must have a Map entity
F-1-5-1	The Map must have a set of predefined spawn-points for the entities to spawn
F-1-6	The game must have a GameEngine entity
F-1-7	The different entities must be able to be loaded and unloaded during run-time
F-2	The game should implement wall collision
F-3	A component framework has to be applied to project
F-3-1	The applied component framework should support multiple classloaders and component versioning

3.2 Non-functional Requirements

ID	Description
NF-1	The functionality of the game, must be implemented in Java
NF-2	The loading and unloading of components should be done within 2 seconds
NF-3	The path finding of the enemy entity should be calculated every 500 ms
NF-4	The handling of component loading will be done through the OSGi framework

4 Analysis

The analysis process has been undergone through various in-group meetings, meetings consisting of all project-groups, and as well with the project instructor. Those meetings gave an overview of the general purpose of developing our game, covering various areas of implementation, in regards to used framework, used entities, and interaction between those. The remainder of this section will therefore be containing a reflection of the above statements, such as related work, differences, purpose and other relevant analysis-based considerations

4.1 Interfaces

Based on the requirements, it is possible to derive some abstractions about the system in the form of interfaces. By doing this, it is possible to create uniform entities, as they have to implement the abstract methods.

4.1.1 IGamePlugin

This is the interface that will handle which entities exist in the world. The interface provides two methods, each with their own responsibilities. Both methods will require instances of the class that holds data about the world, and the one which holds data for the current events in the game. The first provided method will be implemented to make sure the entity is properly created and added to the game. The other will remove the entity from the classes that holds it's data.

4.1.2 IEntityProcessing

The interface will be used for handling the movement of entities on the map. Each entity with its parts will be moved together and processed by the method provided by this interface.

4.1.3 IPostEntityProcessing

This interface will be used for processing events like collisions. This interface will provide a method that will be used for processing the given events, with respects to the conditions the given event must respect. This interface will be especially helpful for handling attacks in the game, or any case where i could be useful to check something after the values of the different entities have been computed in the game loop.

4.1.4 IWeapon

This interface will be used for handling the weapons of the game. This interface will provide methods which must be used to implement the damage models and the attributes of the given weapon. The game will support different types of weapons, and their handling will differ based on the type. If a weapon should have some special abilities, the interface will also provide a method to set this up properly.

4.1.5 IPathFinder

This interface will be used for handling the path-finding abilities of the enemy entity. The methods this interface will provide will be subsidized by an algorithmic implementation that will be the foundation for our path-finding capacities. This will essentially make sure that the enemy entity has the right means to find its way towards the player, given

the certain circumstances. This can be inhibited by any perks that the player may have equipped, or enhanced based on the circumstances.

4.2 Components

This section will describe the required components, their functionality and purpose in the game.

4.2.1 Core

The game engine is made from scratch by the group, using a game development framework as a foundation. Every object made in the engine is to be considered an entity, that implements one or more interfaces. The engine glues the different pieces together, and is responsible for adding or removing them to the game world, depending on the given use case. Simply put, the engine will be the brain behind the handling of game logic, visuals and the audio of the game. In our i

4.2.2 Entities

Weapon The weapon is an entity that is used to deal damage to either the player or enemy. The weapon entity does not function on its own, but has to be in the possession of an entity, to fulfill its purpose. Weapons spawn on the map, where it is possible for the user to pick them up, and control them. Different weapons may have different attributes such as damage impact, which make them suitable for different scenarios. Apart from spawning on the map, they could also spawn in possession of an enemy.

Player The player is an entity that is controlled by the user, and functions as the main character of the game. The player's objective is to beat the game by exploring the map and defeating the enemies and obstacles in his path. The player has the ability to pick up different weapons and various other items. The player can then defeat enemies with the weapons found along the way, and use different items to interact with the environment, and strengthen the player further. The player has health points, representing the players remaining life, which if reaches zero, means death / game over.

Enemy The enemy is an AI-driven entity that is the main obstacle for player, whose purpose is killing the player. The enemies spawn on different parts of the map, and can use weapons to inflict damage upon the player. As enemies are driven by AI, they can calculate and find the most suitable path towards the player. If they are within a certain radius of the player, they will begin attacking the player. Enemies have their own health points and can be defeated by the player, if their HP drops to zero.

Map The map is a component which functions as the accessible area for the entities, in which the game takes place. It can be seen as the most centralized element among all the entities, as the map is independent from other entities. The map will accommodate spawn points, where it is possibly for entities like enemies, weapons and the player.

Free movement on the map is possible, but restriction blocks will be placed, to disallow the entities to move through certain parts of the map. Furthermore it would not be possible to travel through the borders of the map. The map will not have rooms where different views will be presented, but instead serve as one big room.

4.3 Game play (use cases)

In our project, we use use cases in a way to effectively describe the requirements that are crucial to our project in order for it to function properly in accordance with functional and non-functional requirements. This approach is knows as being agile, by getting closer and upfront with what really is required for a given project to be implemented. This approach makes sure that both developers and users understand what is needed and what is nice to have. For instance, using user stories is a very effective way of eliciting requirements and filtering out unneeded elements and focus on what really is needed.

4.3.1 User stories

Based on the section "Top-down genre features", we have performed a noun-verb analysis, to help the group get a better overview of the project. It will help the group to create use cases, and figure out which functional requirements that the group wants to focus on.

Our current user stories are based on essential units that are main actors in the game. Those are respectively the "Player", and "Enemy". The MoScoW prioritization technique is then used further down to show in correlation with user stories, which requirements are the most essential and in need to be implemented first.

Verb/Noun-analysis The group made a verb/noun-analysis, to clarify the different functionalities and aspects, that should be implemented in the game. The analysis was made one the Top-down genre features, which states the general characteristics of the genre. This helps the group to benefit from the points, and form something that later on can be formed into requirements. Each row consists of a relevant relation between a noun and a verb, found through the analysis.

Nouns	**Verbs**
Missions	Has to undergo
Bosses	Confront
Storyline	Develop
Choice	Which missions to execute
Player	Progresses
Goals	Achieve by fending off enemies
Rewards	
Weapons	Get through several levels and challenges
Enemies	Develop and require greater resistance
Objectives	Executed in a sequence of time
Consequences	Affect the general outcome

To help the group create detailed use cases, the group decided to come up with some user stories, based around some different entities in the game, as a result from the verb/noun-analysis.

Player User story for the player.

ID	User story
1.1	As a player, I want to face challenges and enemies throughout the game, so that I get entertained and do not complete the game right away
1.2	As a player, I want the enemies to implement artificial intelligence in their path-finding algorithm, so that I experience realistic movements and get challenged further
1.3	As a player, I want to be able to use weapons, so that I can fight off the enemies I encounter throughout the game
1.4	As a player, I want the different entities in the game to gradually change, so that I get entertained, and do not find the game uniformed
1.5	As a player, I want to be able to choose my moves and decisions, so that I get to explore and make a mark on my completion
1.6	As a player, I want to have a point-system in the game, so that I can strive to better my performance, and use the game competitively with my friends

4.3.2 Use case list

ID	Use case	Description
1.1	Move	The player uses keystrokes in order to move the character around in-game on the map. This would be the W,A,S and D keys
1.2	Interact	The player uses the E key to interact with an object on the map which the player is currently located next to
1.3	Use weapon	The player uses the mouse key to attack with a weapon in the direction that the players character is pointing in.
1.4	Discard Weapon	The player uses the G key to drop the current equipped weapon.

4.3.3 Use case diagram

To illustrate and get an overview of the player's interactions with the game, the group chose to make a use case diagram. The diagram also helped us with defining some of the functional requirements.

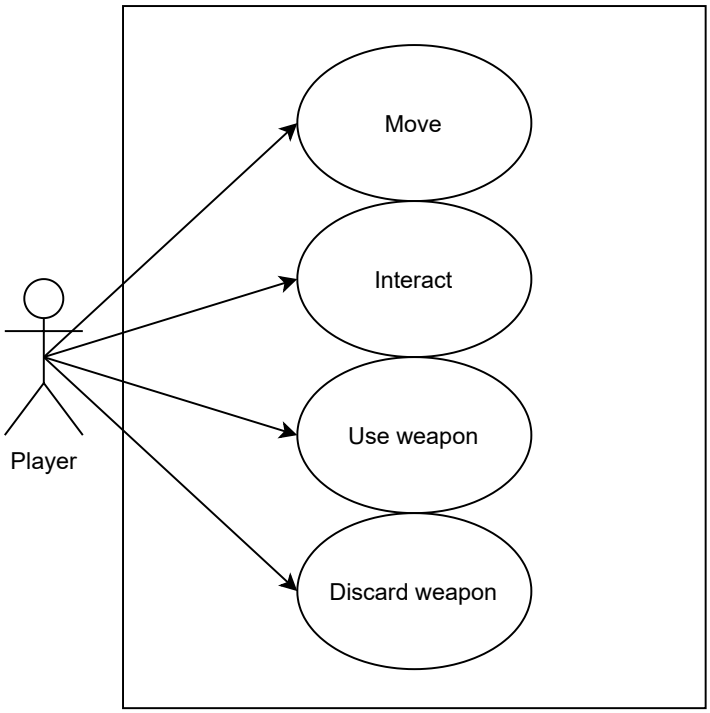


Figure 2: Use case diagram

4.3.4 Detailed Use Cases

Use Case	Interact
Primary Actor	The player
Secondary Actor	None
Description	Interacting with objects
Preconditions	The player is located next to an object on the map
Basic flow	• The player is located next to an object • The player use the E key to interact with the object • The interaction happens
Postconditions	The object has been interacted with

Table 1: Use case: interact

Use Case	Use weapon
Primary Actor	The player
Secondary Actor	None
Description	The player can use equipped weapons with the press of a button
Preconditions	The player has a weapon equipped
Basic flow	• The use case starts when the user presses the attack button • The weapons' attack animation will be displayed
Postconditions	The weapon has lost some if its ammunition

Table 2: Use case: use weapon

4.4 Prioritizing requirements

4.4.1 MoSCoW

ID	Name	Description	Priority
F-1-1	Player	The player must have a character, which is used to interact with the game world	**M**
F-1-1-1	Player Movement	The player must be able to move around the game world	**M**
F-1-1-2	Weapon Interaction	The player should be able to pick up a weapon from the game world	**S**
F-1-1-3	Discard Weapon	The player should be able to drop an equipped weapon onto the game world	**C**
F-1-1-4	Weapon Interactions	The player should be able carry several weapons at the same time	**C**
F-1-1-3	Enemy Damage	The player should be able to deal damage to the enemies in the game	**S**
F-1-1-6	NPCs Interaction	The player should be able interact with the NPCs on the map	**C**
F-1-2	Enemies	The player must be able to encounter enemies	**M**
F-1-2-1	Player Damage	The enemies should be able to be damaged by the player	**S**
F-1-2-2	Enemy AI	The enemies must have a simple kind of artificial Intelligence	**M**
F-1-2-3	Path-finding algorithm	The enemies should implement the A-Star algorithm for path-finding	**S**
F-1-3	Weapons	The player should be able to use weapons	**S**

F-1-3-1	Weapons Stats	The player should gain benefits by using weapons	**S**
F-1-3-2	Weapons	The player should be able to discard of an equipped weapon	**S**
F-1-3-3	Weapons ammunition	The weapons should have a finite number of bullets to shoot	**C**
F-1-4	NPC	The game could implement NPCs as an entity	**C**
F-1-5	Map	The player must be able to interact with the game world	**M**
F-1-5-1	Spawn-points	The map could have a set of predefined spawning-points for the entities to spawn on	**C**
F-1-6	Game Engine	The game must be run through an engine	**M**
F-1-7	Components	The game components must be able to load and unload during run-time	**M**
F-2	Wall Collision	The player should not be able to move through in-game objects	**S**
F-3	Component Based	The game must be component based	**M**
F-3-1	Component Based	Modules should be loaded on and off dynamically while the game is running	**M**

4.5 The object model

To better understand how the system will be created, it is important to diagram the object relations to visualize the practicalities of the system. It must be noticed that the object diagram functions as a snapshot of the running system, and the values which the objects hold.

4.6 Communication between components

Sequence diagrams here

The communication between components is a very important aspect of a component-based system. It is important to agree and have a vision on how the structure of the different components will be, to secure a logical bond between them.

The overall concept in this project that is applicable for the communication between the components, is that the independent entities mostly have a SPI(Service Provider Interface), to handle its incoming communication. Furthermore the project is set up in NetBeans OSGI system, which allows for smoother and more efficient communication between the components.

The group has developed a sequence diagram of two main usecases, which in practice has to communicate to fulfill its purpose. The sequence diagrams gives a general insight in how the communication and handling of data between the different compontents

4.7 Planning Poker

Planning Poker is the process of estimating the required hours to implement the different aspects of the system. The group made the Planning Poker to have a basis for the implementation-plan, where the number of weeks spend on the different aspects would be planned. Furthermore, the exercise caused the group to bring up the disagreements about the implementation, which where discussed and contributed to the group having a clearer vision and expectation to the implementation-phase.

ID	**Hours**
F-1	89
F-1-1	13
F-1-1-1	5
F-1-1-2	5
F-1-1-3	5
F-1-2	21
F-1-2-1	5
F-2-2	21
F-1-3	13
F-1-3-1	5
F-1-3-2	8
F-1-4	21
F-1-5	15
F-1-6	8
F-2	5
F-3	8
F-3-1	8

Table 4: Result from Planning Poker session

After having the Planning Poker made, the group planned the implementation more specifically throughout the following weeks. This would be used as a work-plan, describing what would be implemented when, and how much time it would take. This plan can be seen in the appendix. [Appendix : A]

5 Design

Design is used to describe the design choices in alignment with the initial requirements analysis alongside the architectural choices. Several different design patterns can be used to structures and utilize the design of the application. Several different design patterns and principles can be utilized, such as MVC, DAO, SOLID and dependency injection. In our application, we have chosen to implement the SOLID principle to handle the design structure of the game.

In this section we will go over the Component Framework, which generally known is a Java programming language that utilizes the component model to dynamically load and unload bundles, modules, services etc. That is done through constructing the component contracts, illustrating their purpose and use. Alongside that, we will briefly go over our class diagram, as an intermediary for the next section, the UML component diagram. The UML component diagram is used to illustrate the component model of the game. The purpose behind the below sections are to help model the implementation and make sure that every aspect of the game has undergone inspection to accommodate the initial requirements.

5.1 Component Framework

The game will use OSGi as a component framework. By using this, it is possible to fulfill the requirement of dynamic loading and unloading of components. The interfaces will primarily be connected by the declarative services API **(Bundecontext API?)** which provides the ability to use dependency injections. Another reason to use this framework is also the support of the Apache Felix Gogo shell, which will make it easier to test the dynamic capabilities of the components.

Declarative services are done via another component model called Dependency injection. It works through using our manifest files, that are XML formatted files found in the manifest folder. There, it specifies the implementation of the given service working as a key-value pair – the key would be the name for the service provider interface, where the value is the name of the implementation file.

SPI	**IGamePlugin**
Operation	`void start(GameData gameData, World world)`
Description	The start method is called when the application starts. This will instantiate the entities.
Parameters	`gameData`- the data contained within the GameData object of the game `world` - the data regarding the entities that are present in the game world.
Preconditions	There is a set of components to be loaded, and the component is not already loaded into the system. The component exists in the registry, or has been registered as an implementation of the interface.
Postconditions	The available components have been loaded into the system, and are active through one or several instances

SPI	**IGamePluginService**
Operation	`void stop(GameData gameData, World world`
Description	The stop method is called when the application is closed. This will remove the entities.
Parameters	`gameData` - the data contained within the GameData object of the game `world` - the data regarding the entities that are present in the game world.
Preconditions	The plugin has been loaded into the application.
Postconditions	The plugin has been unloaded, and removed from the game world.

SPI	**IEntityProcessingService**
Operation	`void process(GameData gameData, World world)`
Description	In this method the entities in the game should be processed and updated. No heavy calculations should performed here.
Parameters	`gameData` - the data contained within the `GameData` object of the game. `world` - the data regarding the entities that are present in the game world.
Preconditions	Data has not been processed
Postconditions	The entities in the world have been updated (processed) in accordance with the game data.

SPI	**IPostEntityProcessingService**
Operation	`void process(GameData gameData, World world)`
Description	In this method the entities should be processed. In this method calculations should be done on the entites where the state of the data of the entities are relevant.
Parameters	`gameData` - the data contained within the `GameData` object of the game. `world` - the data regarding the entities that are present in the game world.
Preconditions	The entities in the game have been processed.
Postconditions	The entities have been processed.

SPI	**IWeaponSPI**
Operation	`Weapon createWeapon()`
Description	When this method is called, a weapon is created.
Parameters	
Preconditions	An implementation of the WeaponSPI has been installed in the system.
Postconditions	A weapon has been acquired.

SPI	**IWeaponSPI**
Operation	`void attack(Weapon weapon)`
Description	The attack method is called, when a collision between the weapon and a given entity is detected.
Parameters	`weapon` - is the weapon being utilized.
Preconditions	The weapon exists in the game world.
Postconditions	The weapon has generated its attack in the game world.

SPI	**IWeaponSPI**
Operation	`void destroyWeapon(Weapon weapon)`
Description	The weapon is destroyed when this method is called.
Parameters	`weapon` - is the weapon to be destroyed.
Preconditions	The weapon exists in the game world.
Postconditions	The weapon has been removed from the game world.

SPI	**IPathFinderSPI**
Operation	`float track(Entity from, Entity to, World world)`
Description	Service Provided Interface between an entity, to find the path to another entity. Returns a float with the correct direction in radians.
Parameters	`from`, `to` - Enties in the game. `World` - Data about the world of the game.
Preconditions	The `from` entity does not already have a direction.
Postconditions	The Entity `from` is pointed in the direction of `to`.

5.2 OSGi Framework

The OSGi framework (Open Services Gateway initiative) is a platform used for implementing a component model. Applications/components can come in the form of bundles, which introduces the whiteboard model Bundlecontext API. It is a whiteboard model, as it works through registering services and implementing them through an activator that then hooks into the life-circle of the OSGi run time container found in the application. In the activator file, a start method is seen where a service gets registered through calling the "registerService" method. It points towards the implementation of the desired service that we want to be implemented.

Several states are defined for bundles after them being installed, seen as below

–image of bundle states–

These states determine how the installed bundles can be run in the runtime framework. A bundle installed is at a resolved states, meaning it is ready for deployment into the runtime-container, that is invoked through the start-method in the activator class. From there it can be stopped and moved back to its resolved state, or completely uninstalled if needed.

5.3 Class Diagram

5.3.1 Core Module

Figure 3: Class diagram for the Core package

5.3.2 Common Module

Data package The data package is located in the common package

Figure 4: The Data package

EntityPart package The entitypart package contains data classes, that can be registed to an instance of `Entity`.

Figure 5: The EntityParts package

Services The `Services` package contains the public interfaces on which implementation of the components can depend.

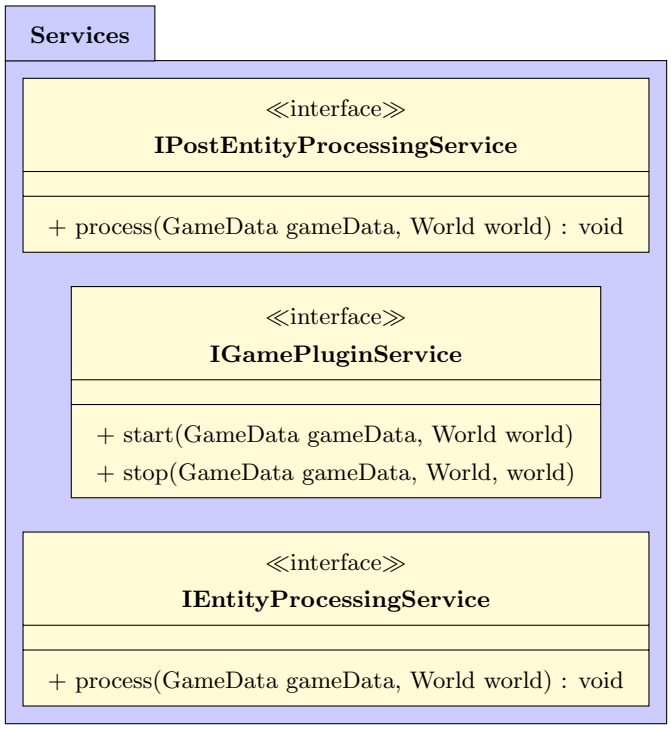
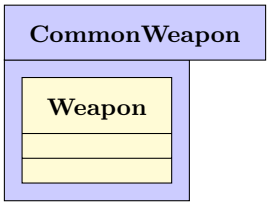


Figure 6: The `Services` in the `Common` package

5.3.3 Common Weapon



5.3.4 Common Player

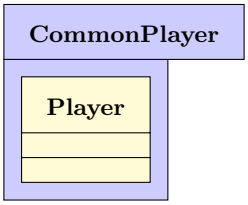


Figure 7:

5.3.5 Common AI

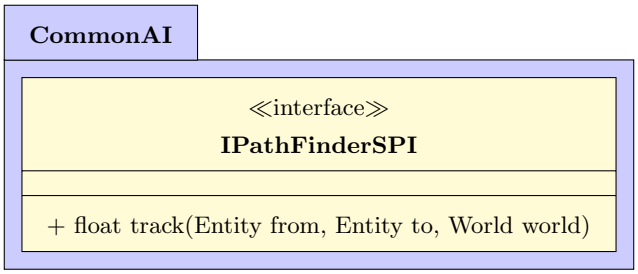


Figure 8:

5.3.6 Common Map

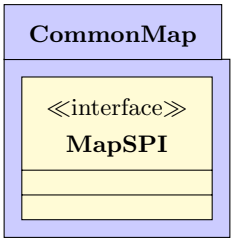
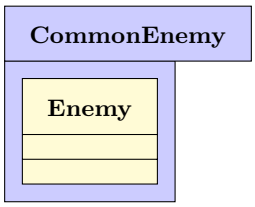


Figure 9:

5.3.7 CommonEnemy



5.3.8 Enemy

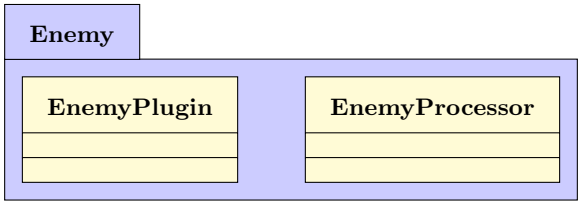


Figure 10:

5.3.9 Player

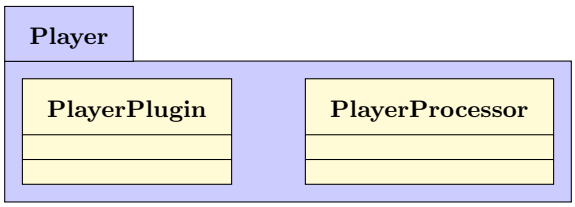


Figure 11:

5.3.10 AI

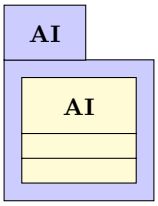


Figure 12:

5.3.11 Collision

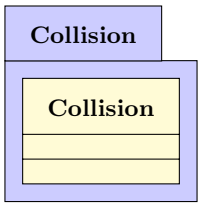


Figure 13:

5.3.12 Map

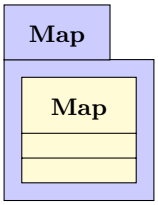


Figure 14:

5.4 Component Diagram

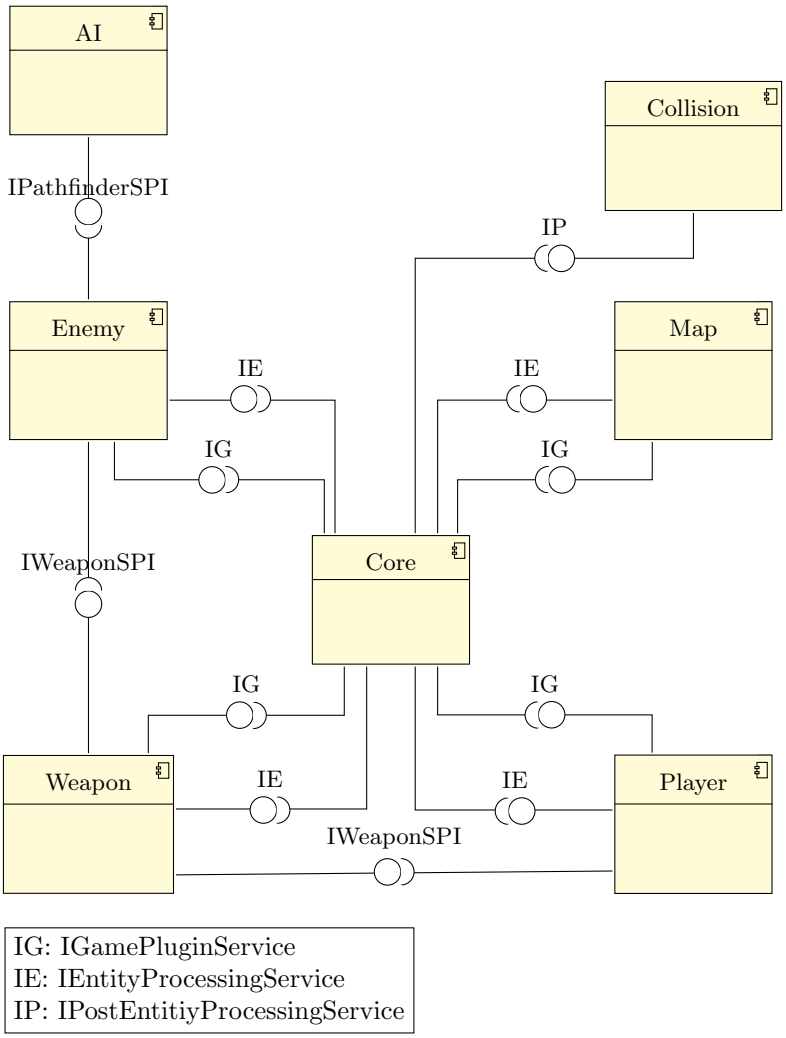


Figure 15:

5.5 Artificial intelligence

It has been decided that at least one entity in the game, has to implement artificial intelligence. The group chose to implement AI through the enemy, which will have a path-finding algorithm, for following and attacking the player around the map. AI is a broad spectrum covering a lot of different algorithms - and the path finding algorithm being implemented through the enemy, is the A-star algorithm

5.5.1 A-star search Algorithm

The A-star search algorithm is a widely used algorithm in games, where entities have to find the the shortest way to points around obstacles. One of the strongest characteristics behind the A-star algorithm, is its ability to spot a problem before it approaches it. This is possible due to the fact that the algorithm makes its path based on a combination between the best next step, and the overall shortest direct distance to the goal. It

does not search blindly, but makes a concentrated exploration, which branches from the "so far" best path. The A-star algorithm therefore finds the overall best path to the goal. The downside to this algorithm is its time and space for performance. The A-star algorithm has to explore and save several paths, which takes up space and time. But these disadvantages will never be noticeable in the context of a simple game, as the one being developed through this project. It is therefore a big advantage to use this algorithm in this context.

6 Implementation

7 Test

8 Discussion

9 Conclusion

References

[Wik21a] Wikipedia. *Hotline Miami — Wikipedia, The Free Encyclopedia*. (accessed 10.03.21). 2021. URL: https://en.wikipedia.org/wiki/Hotline_Miami.

[Sco] Scoring. *Scoring — Hotline Miami Wiki*. URL: https://hotlinemiami.fandom.com/wiki/Scoring. (accessed: 08.04.21).

[Wik21b] Wikipedia. *The Binding of Isaac (video game) — Wikipedia, The Free Encyclopedia*. (accessed 11.03.21). 2021. URL: https://en.wikipedia.org/wiki/The_Binding_of_Isaac_(video_game).

A Implementeringsplan

Medlemmer		
Mohamed, Yonus, Joakim, Armin, Ammar, Peter, Jonas		**Forventet tid brugt**

Implementering	MoSCoW	15	16	17	18	19	20	21
Implementering af Map	M							
Implementering af spawnpoints	M							
Implementering af Player	M							
Player bevægelse	M							
Player våben benyttelse	S							
Implementering af Enemy	M							
Implementering af AI pathfinding	M							
Enemy våben benyttelse	S							
Implementering af Healthpoint-system	S							
Implementering af Våben	S							
Implementering af affyring	S							
Implementering af collision system	S							
Implementering af NPCs	C							
Player til NPC interaktion	C							

B Combined class diagram

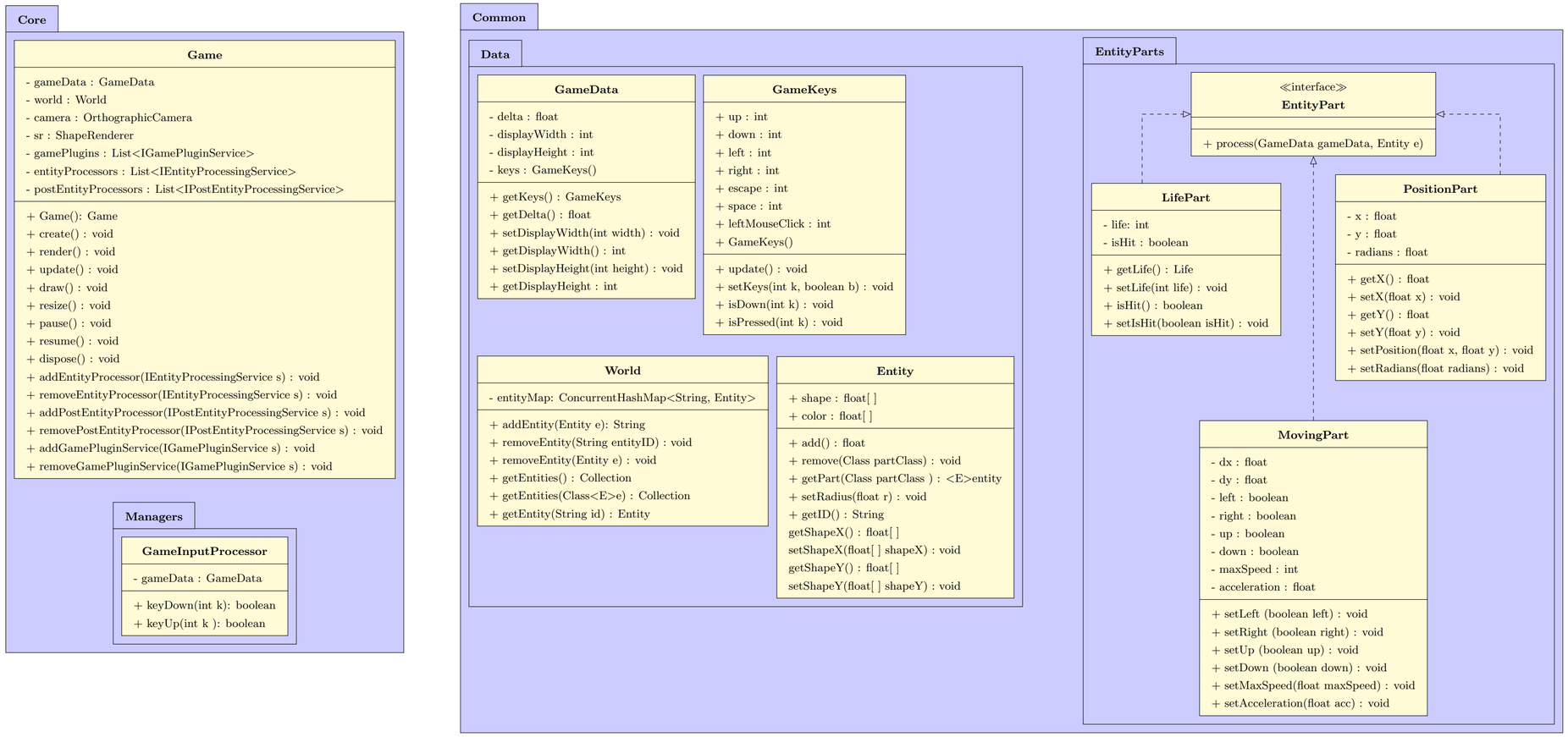


Figure 16: