Internet Game Design and Development

CI328 - Report

harrys pitsillides

2017

Contents

[Game summary 2](#_Toc501663845)

[Objective 2](#_Toc501663846)

[Rules 2](#_Toc501663847)

[Game play 2](#_Toc501663848)

[Implementation 3](#_Toc501663849)

[Introduction 3](#_Toc501663850)

[Programming Language 3](#_Toc501663851)

[Game World 4](#_Toc501663852)

[Architectural Pattern 4](#_Toc501663853)

[Physics Engine and collisions 4](#_Toc501663854)

[AI 5](#_Toc501663855)

[Effects 6](#_Toc501663856)

[Mobile device compatibility 7](#_Toc501663857)

[Conclusion 7](#_Toc501663858)

[Run application 7](#_Toc501663859)

[References 8](#_Toc501663860)

[Appendix 9](#_Toc501663861)

# Game summary

## Objective

In this game currently, there are two stages a user can play however, both stages have as a winning condition that a number of enemies that must be killed before a stage is considered ‘cleared’. This game attempts to stay true to classic side scrollers where the game-winning condition was not so visible but obvious, therefor it is not indicated anywhere trying to mimic this attribute.

## Rules

The user has a set amount of health depending on the difficulty, ones the players life hits zero the player loses. The user has a control of a tank that can fire a bullet once every 1.5 seconds regardless of difficulty, the bullet can vary on shoot angle according to the user input, however the speed is dependent on the tank choice. The user can move left or right to hunt or avoid the waves of enemy tanks.

## Game play

The user starts of at a main menu screen where they can either start a new game with the default settings or they can navigate into preferences and set up any preferences they have such as, difficulty, tank and stage. It is important to note that all of this affect the game in some way. As an example, difficulty affects enemy tanks ‘prediction’ capabilities on where to shoot and how much damage they deal furthermore it affects the users damage and health. The tank selection affects the range the user tank can fire at the speed it can move at and the damage the user tank can deal. As a note here, each tank uses different kind of bullets that deal more or less damage, this is also true for the enemy tanks, different tanks differ in attributes. Lastly the level selection as seen in the objective section minorly affects the hidden winning condition for example the ‘Candy Level’ has less enemies to kill as a winning condition. However random disaster (missiles that damage the user falling from the sky) occur a lot more often while support crates that heal you, less frequently.

# Implementation

## Introduction

The game attempts to mimic a classic 2D shooter side-scroller. Following the classic directive that the screen will move forward according to the speed and direction of the player character, and can also scroll backwards to previously visited parts of a stage. In addition the game is defined by ‘multiple distinct levels’ and extends beyond the single static screen, which are part of for horizontal side scroller1, 2.

Additionally, the game offers a selection from five different tank selections (see Appendix Figure 1.), two different distinct levels with different environment’s and enemies approaching you from all sides to increase the fun and difficulty. Furthermore this gives the user the ability to shoot in all directions something that was considered a breakthrough in the shooter side-scroller genre and defines it, as first seen in the 1985 release of Thexder3.

## Programming Language

The main structure of the game is written in Typescript4 it was chosen as an alternative to JavaScript. Typescript is a superset of JavaScript, it is an open source library developed by Microsoft. It offers a strict syntax and optionally adds static typing to the language. It provides the static typing through type annotations thus enabling type checking at compile time5, 6. Moving beyond the point of what Typescript is it is necessary to explain why it was chosen over JavaScript, even though JavaScript is standardized through the ECMAScript standards, it lacks on all browser compatibility, something that Typescript provides by compiling to older ECMAScript targets. Additionally, as seen above providing type support which will most likely never be standardised in the ECMAScript’s due to the interpreted nature of JavaScript something that will be discussed in more detail further on. Typescripts philosophy was that it would be possible to practically change the files extension from .js to .ts and start using the language. This is implemented by having TypeScript compile to readable JavaScript. Moving on to the most significant difference, as discussed in multiple occasions, JavaScript is a dynamically typed language, which means it does not know the type until run-time, which might be to late. TypeScript improves on that by providing types, thus removing bugs that can be caused by false assumptions of some variables. Unlike most high-level languages such as Java, TypeScript uses type inference, what this means that, as seen in the code snippet below, both ***num*** and ***typeNum*** are of the same type “number”. The type is simply inferred from its use even if not explicitly defined.



As said by Lodewij Bogaards CTO at StackState in an answer at StackOverflow7 that was in its documentation prior to them being decapitate8 “A Large Scale Study of Programming Languages and Code Quality in Github suggests that that “statically typed languages in general are less defect prone than the dynamic types, and that strong typing is better than weak typing in the same regard9.””

In addition to that the same paper has resulted that TypeScript is less error prone to JavaScript.

## Game World

### Architectural Pattern

This game follows the Entity - Component – System (ECS), architectural pattern. ECS is true to the composition over inheritance principle. It is a principle which dictates that all classes should have polymorphic behaviour and allow code reuse, instead by following the inheritance principle, by instead having instances of other classes that contain the end behaviour/functionality10. This allows us to define every game object in the world as an entity in our case the tanks, bullets, power ups. Then we can use these entities to add or remove functionality (as seen in figure 2 code snippet) by loading or removing components. What this means is that in theory entity behaviour should be able to be changed at runtime. This approach is supposed to fix the ambiguity problems of inheritance hierarchies which introduces unnecessary coupling in what can be described by Leonid Mihkajilov as “fragile base class problem”11. In the case of this application the system is implemented as a sudo-system, this means there is no dedicated system that performs any global actions on the entity, instead on the entities base class, in its own update function which is called on the game update phase it loops through every component and runs its update function (see appendix figure 3).

### Physics Engine and collisions

Phaser offers 3 different physics engines, arcade, ninja and p2.js it is worth mentioning that Phaser ships with the arcade physics engine, ninja and p2.js are external libraries. Arcade Physics is for High-speed AABB (axis aligned bounding box) collision. Thus, this is limited for what this application aims for as the tanks should be able to rotate, apply force restrictions (such as projectile and tank velocity and drag), and some levels will not be comprised only from rectangles. Moving on to ninja physics, it can handle rotation and can do slopes and complex tiles but is more limited on applying velocity, drag etc to game objects and if far less documented/used than the P2 engine. Finally, the engine that was chosen for this application, the p2 physics engine. This is a full-body physics engine, it allows polygon support, force constrain, is well documented with plenty of examples and provides support with Tiled12, a game map design application. This however comes with some unwanted effects that should perhaps have been give more of a consideration before the implementation of the game, the game will get considerably heavier (slower) as the P2 engine requires sufficiently more recourses as it is a full body physics engine, which means it must do a lot more calculations and collision detections. Additionally a lot of ready functions that aid in object movement towards other objects (accelerateToObject13, accelrateToPointer13 see appendix figure 4. for implementation) and angle calculations that exist in the arcade engine must be implemented manually. This would require a sufficient level of research in physics and mathematics, something that had increased the development time greatly. Furthermore, even though the P2 engine offers object rotation, it does not however offer a way to limit or enforce the rotation order, which resulted as an example, in the tank being flipped upside down and moving forward (see appendix figure 5. for implementation). As this had to be implemented manually it has produced some unwanted behaviour of some tanks looking like they are bouncing and having a weird movement behaviour. As the issue begins with the physic engine it would be hard and extremely time consuming to fix this behaviour therefor it was left to be as such.

Moving to collision handling, the P2 engine has two ways of handling collisions, one way would be to set collision call-backs on the sprite’s body and the other one is to create collision groups and specify the action that occurs between each collision group then add the sprite to that group. The second one involves a lot more code but gives greater control over the collision event. In addition, it allows sprites to ignore other sprites in this world, which gives us the overlay effect. As in the application there is need for objects to overlay each other without necessary colliding, the second option was chosen. Unfortunately, as mentioned before the second option involves more code, this could appear a bit messier as when declaring the actions that should occur when two collision groups collide, both objects will emit the impact event, this means one of the two objects should have an action of do nothing just collide otherwise the effect would occur twice (appendix figure 6.).

### AI

This application utilizes the finite state machine (FSM) to determine the behaviour of the AI. FSM is a computation model that can be in one of a finite state of states at a given time. The FSM model can respond to external inputs to change from one state to another. It is defined by its states and the conditions to switch from one state to another. However, this posed an interesting challenge that needed to be resolved before moving on. As mentioned prior to this, the application implements the ECS as the architectural pattern therefor it would be necessary to have a middleware between the FSM and the entity that would be targeted. The solution for this issue was to create a component that would control the loading and unloading of states from a given entity (see appendix figure 7.). Additionally to stay true to the single responsibility principle that all classes should have only one reason to change14, an AI Component was created (see appendix figure 8.) to handle the state changing logic. However this AI component violates the open/closed principle, that a class should be open for extension but closed for modification15, perhaps a better alternative would be to allow the AI component to take a “decide function” ,as an example, assuming this wasn’t about a tank but a helicopter AI the decide function would still run in the component but it would run the passed function. In this case it was chosen to be left as such as there is no plan to allow any other form of AI to exist in this application throughout its lifetime. Therefore, simplicity was chosen during the development stages of the project on the developing stages of the AI component.

#### AI States

Moving on to the AI states, the FSM contains five states, Pursing, Seek, Evade, Suicide and Flee.

***Pursing state*** (appendix figure 10.): A Pursuit is the process of following a target aiming to catch it. However, this is not something desirable in this application, what is desired it to pursuit a target and until is within a range of max range minus a set value that allows for the AI the room to make a prediction on the players next position. The pursuit state will work in a way in the same sense as seek but instead of the player itself it will attempt to seek the players future position. What can be done to achieve this is, we can get the seconds necessary to reach its target by dividing the distance between the two targets and the speed. Thus, we can get the position in the next second by getting the current sprite position plus the sprite velocity divided by 1000 (Phaser cycle, one second) and multiplied by the seconds necessary to reach the player. Then the state checks if the position of the player is between the range of the projectile and a difficulty modifier (the greater the difficulty the more precise the aim is) to see if the AI can hit the player.

***Seek state*** (appendix figure 9.): What this state aims to do is to create a realistic movement pattern, the goal is to have a velocity that is limited to a top speed but have a steering force applied to it to ensure the correct movement direction. The direction is extracted by simply substracting the target position and the current position. Phaser provides a function to do that, all physics object body’s have move functions that will update the position of the body by the given amount. Phaser works by updating pixels per second.

***Flee state:*** This state performs the same function as the seek state but instead will move in the opposite direction of the player but still face in the direction of the player. This will give a more “real feel” to the fleeing state as a tank would flee facing its enemy.

***Suicide state*** (appendix figure 11.)***:*** This state will only run if a tank has less than half health and has near allies. What it will do it will move towards the player once close enough it will explode destroying itself and dealing damage to the player.

***Evade state*** (appendix figure 12.): This state will run once the player is closer than what the AI’s range of projectile is. What this state will do it will accelerate in the opposite direction in what the player is facing. Then move close to allow the pursing state condition to be true.

### Effects

In the following application, there are the three following effects, particles, sound, animations. Particles where implemented in the collision component using the Phaser.Emitter class, this class then allows to call a makeParticle function that allows to pass in what sprite would be used.

For the sound effects Phaser provides a function in Phaser.Loader (Phaser.loader.add.audio) which returns an instance of a Phaser.Sound class, which in turn has functions to start and stop the sound. The above have been encapsulated within a static class in this application (appendix figure 13.) to avoid polluting the global Loader object.

Animation effects have been encapsulated within the Layout component class, this made sense during the implementation phase as the Layout component is responsible for setting the correct frame on the sprite. Perhaps again a better approach would be to create a new Animation component as the layout component violates the single responsibility principle14. Even though phaser already provides functions for this, as the application implements the ECS approach, there was however, a need for a bridge between the two. Therefor what it does, is to call the phaser functions within the components exposed functions (appendix figure 14.).

### Mobile device compatibility

One of the main advantages and reasons why Phaser was used, is that HTML5 & JavaScript games can run on both Desktop and mobile devices. The only differences in code to allow this application to run via a mobile device would be to give the user someway to provide input to the game. The game first checks if it is being run from a desktop by using Phaser’s inbuild class that exposes a Boolean value to determine whether it is a mobile or desktop device. If this is a mobile device, the game will create two buttons at the end of the screen to move the tank in the direction pressed (see screen shot appendix figure 15.). To make the tank shoot the game checks if the activePointer is down, this is mobile friendly and will return true if the user is touching the screen.

# Conclusion

Even though this application attempted to stay true and adhere to most good coding principles such as SOLID16 on some occasions they where disregarded for development speed. Additionally, as discussed on the Physics section, perhaps there should have been a deeper investigation and research regards to what engine to use. Because of the P2 engine being used, a lot of states (FSM States) had to have their scope and functionality reconsidered greatly during development, something which led to increased development time and violations in both the ECS and the SOLID principles. This was due to the P2 getters and setters on velocity and position, as this could not be affected directly without a great deal of side effects. Furthermore, as the ECS approach was used this prevented the creation of phaser groups, which allow for better management of recourse release and manipulation. Instead the application directly destroys these sprites, a function that allows Phaser to indicate that a sprite is ready for garbage collection, instead of using Phaser suggested method of calling the kill function on a sprite and then respawn it at the location. This approach would allow to limit how many sprites are recreated at each time with greater control thus reducing the number of active objects and avoiding memory leaks. Instead the application follows the manual approach which in addition to the issues mentioned above sometimes can cause Phaser to use a sprite that has not been yet reclaimed by garbage collection but has a null body, which leads to some unexpected behaviour such as sprites not appearing, not calling the on-collision events and throwing an error later. This was detected at a very late stage in the development and implementing this approach would require a great deal of code refactoring, instead a messier approach was chosen of if statements and try catch blocks, something that increase the length of the code and reduces readability and perhaps still allowing this behaviour to appear. Finally, the game runs greatly on all devices both desktop and mobile, if the game is launched in chrome. Unfortunately, on Firefox the game suffers performance issues, initial investigation has revealed that this is another issue caused by the great number of collision events fired by the P2 engine.

# Run application

The application can be launched [here](https://harrypi.github.io/CI328WebGame/) (https://harrypi.github.io/CI328WebGame/)

# References

1. Whitehead PJ. Game genres: Shmups. UC Santa Cruiz2007.

2. History of computing: Video games - golden age [Internet].2017 [accessed Available from: <http://www.thocp.net/software/games/golden_age.htm#MoonPatrol>.

3. The leif ericson awards - ign. 2017.

4. Typescript - javascript that scales. [Internet].2017 [accessed Available from: <https://www.typescriptlang.org/>.

5. Type inference â· typescript [Internet].2017 [accessed Available from: <https://www.typescriptlang.org/docs/handbook/type-inference.html>.

6. Hejlsberg A. 3.11.7 type inference [Internet]. <https://github.com/:2012> [accessed 2017 20 December].

7. What is typescript and why would i use it in place of javascript? [Internet].2017 [accessed Available from: <https://stackoverflow.com/questions/12694530/what-is-typescript-and-why-would-i-use-it-in-place-of-javascript/35048303>.

8. Documentation [Internet].2017 [accessed Available from: <https://stackoverflow.com/documentation>.

9. Baishakhi Ray DP, Premkumar Devanbu, Vladimir Filkov. A large-scale study of programming languages and code quality in github. 2017.

10. Knoernschild K. Java design: Objects, uml, and process : 1.1.5 composite reuse principle (crp). Addison-Wesley Inc; 2002.

11. Leonid M, Emil S. A study of the fragile base class problem1998.

12. Lindeijer T. Tiled map editor [Internet].2017 [accessed Available from: <http://www.mapeditor.org/>.

13. Phaser ce class: Arcade [Internet].2017 [accessed Available from: <https://photonstorm.github.io/phaser-ce/Phaser.Physics.Arcade.html#accelerateToObject>.

14. Martin R. Agile software development, principles, patterns, and practices: Patience Hall.

15. Meyer B. Object-oriented software construction: Prentice Hall; 1988.

16. Getting a solid start. Gotham  Ruby; 2013; Duke University.

# Appendix



Figure 1. Tank selection Screen

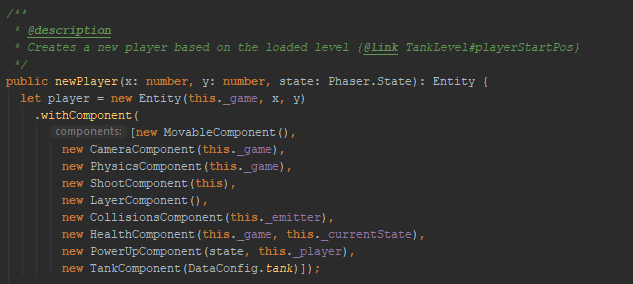


Figure 2. Code snippet of adding components to an entity

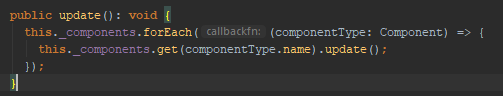


Figure 3. Entities update function gets all components and calls their update function

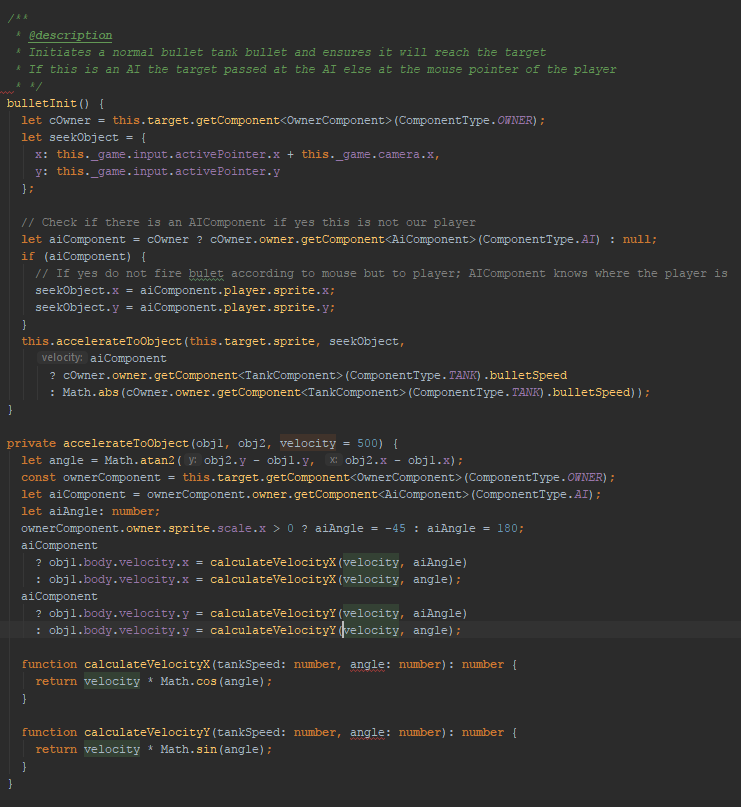


Figure 4. Code snippet of how a bullet detects if it is a mouse click thus accelerating to pointer or accelerating towards an object (tank)

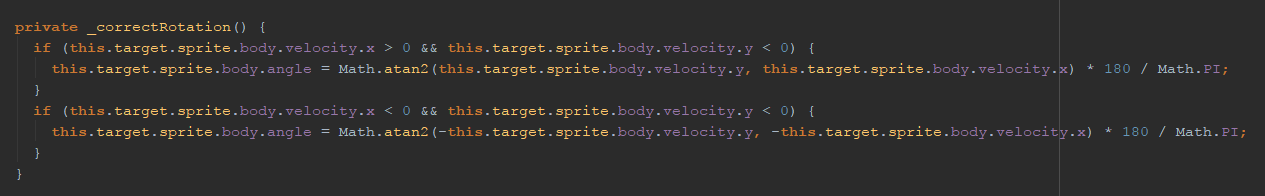


Figure 5. Keep the tank rotation at the correct angle code snippet

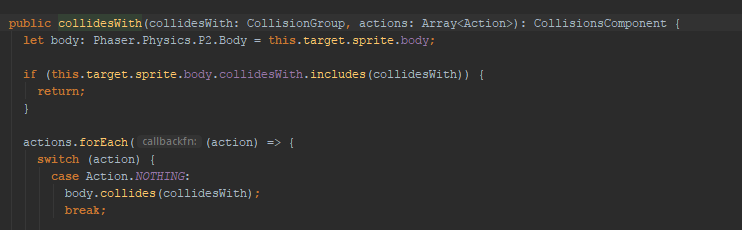


Figure 6. Action nothing when colliding

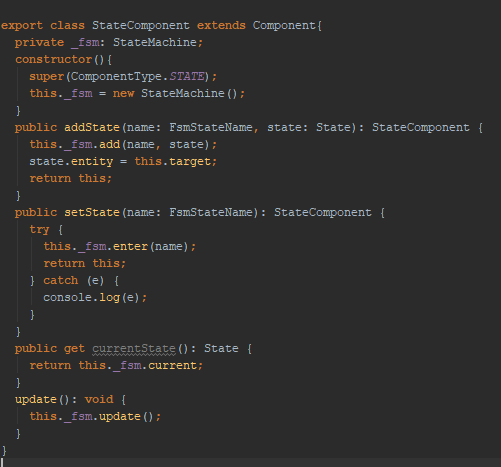


Figure 7. Middleware between ECS and FSM

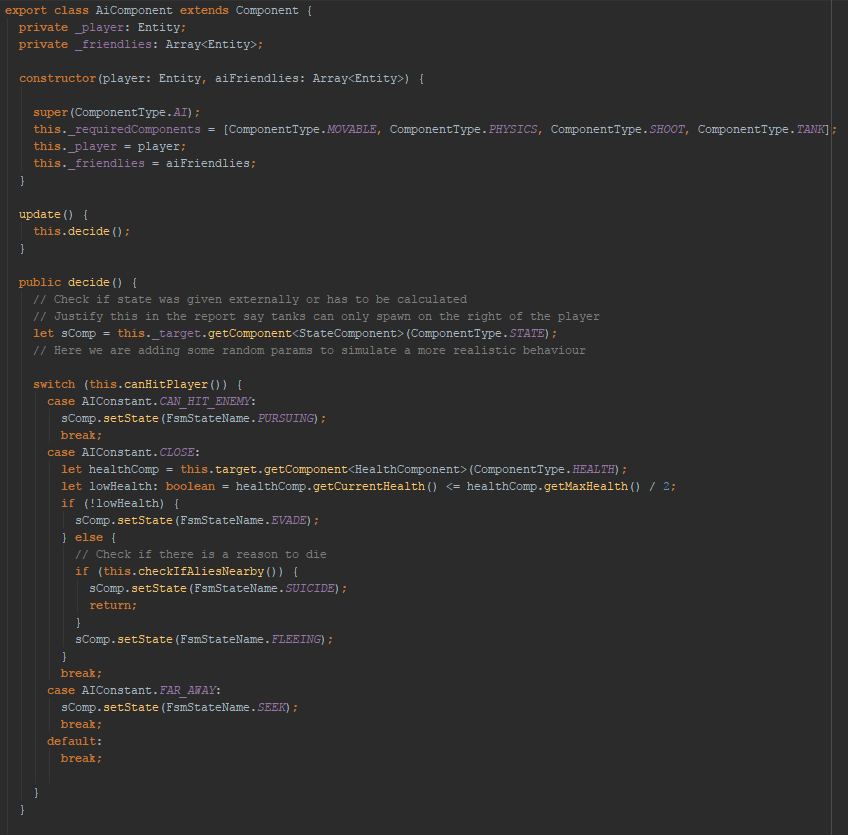


Figure 8. AI component and state changes

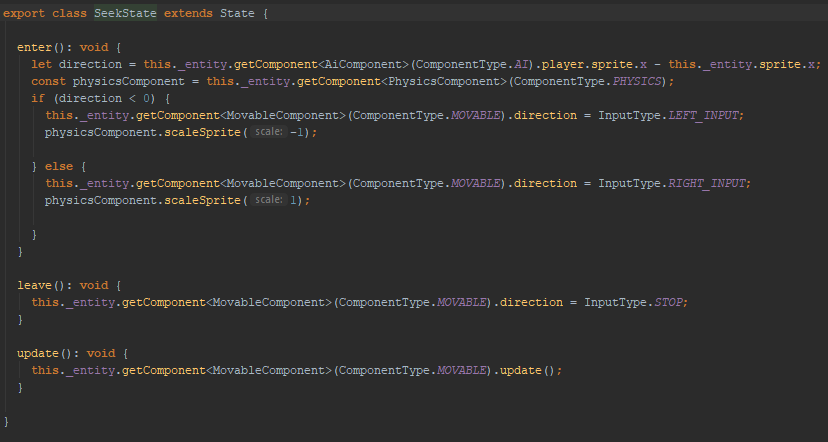


Figure 9. Implementation of seek state

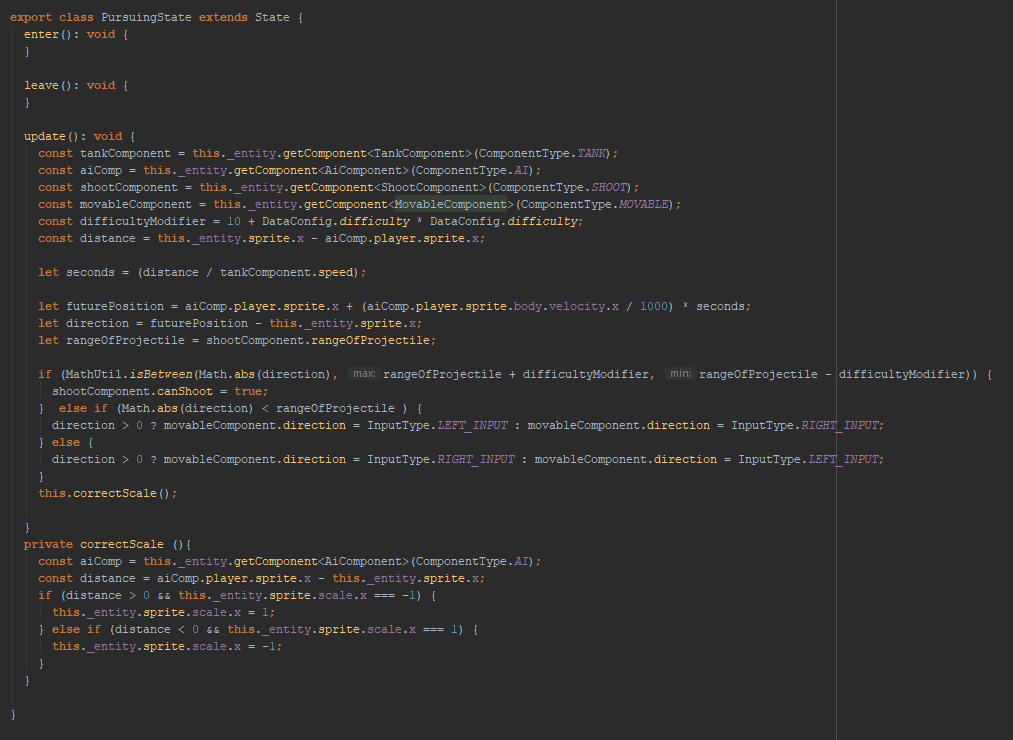


Figure 10. Pursing state code implementation.

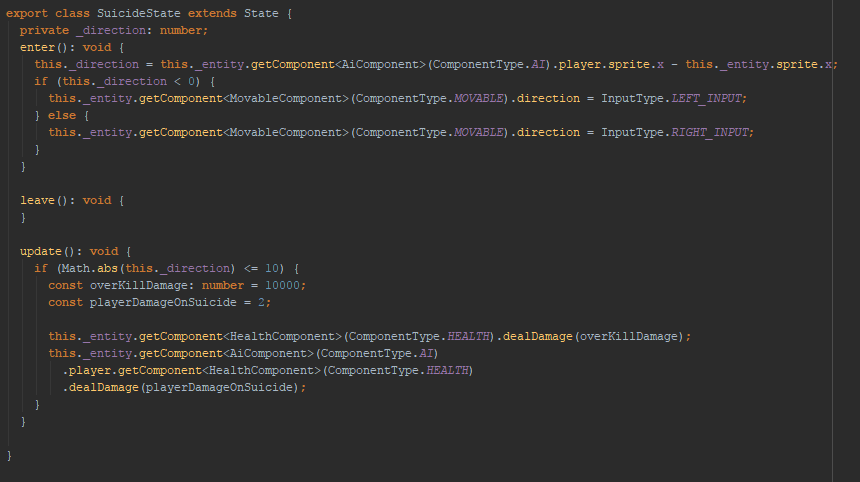


Figure 11. Suicide state code implementation

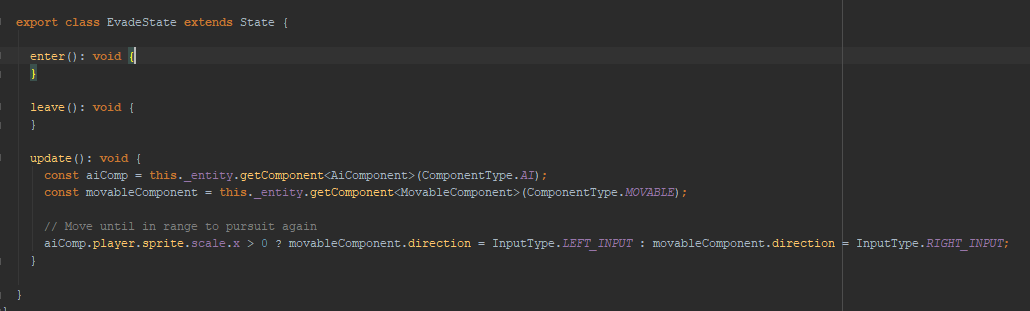


Figure 12. Flee state code implementation

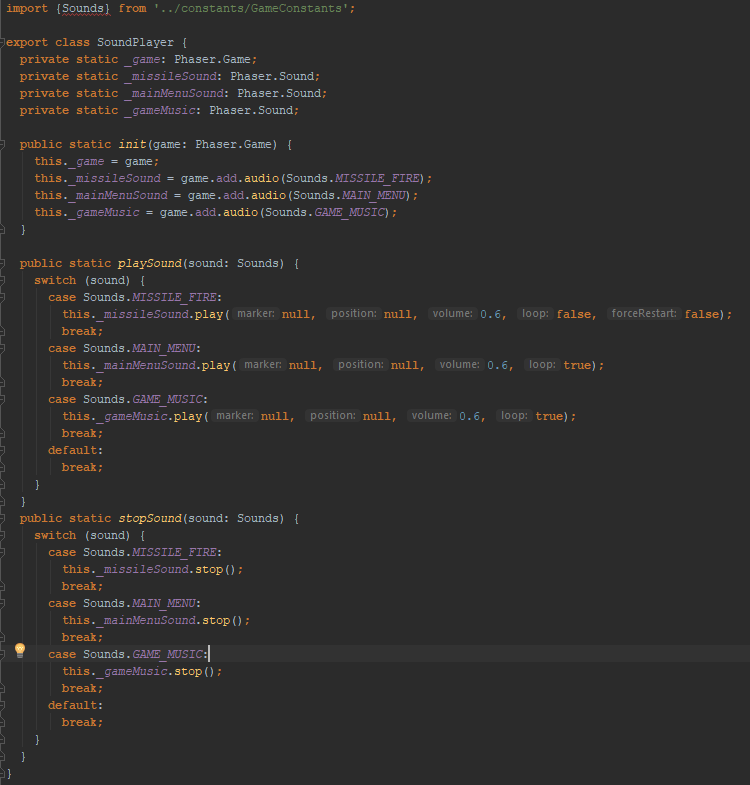


Figure 13. SoundPlayer class code implementation

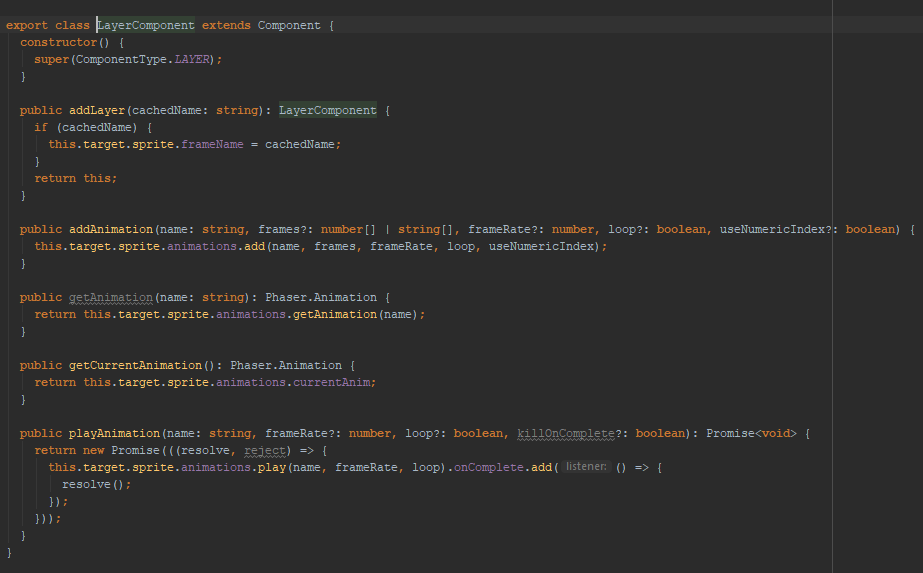


Figure 14. Layout Component implementation.



Figure 15. Mobile phone screen shot