|  |
| --- |
| Multiplatform Endless Running and Fighting Game |
| A-Level Computer Science NEA Project |

|  |
| --- |
| Sergi Lange-Soler  2022 |

# Table of Contents

[1. Table of Contents 1](#_Toc96917893)

[2. Analysis 5](#_Toc96917894)

[2.1. Description 5](#_Toc96917895)

[2.1.1. Gameplay 5](#_Toc96917896)

[2.1.2. Controls 6](#_Toc96917897)

[2.1.3. Theme 7](#_Toc96917898)

[2.2. End User 7](#_Toc96917899)

[2.3. Research 8](#_Toc96917900)

[2.3.1. Shadow Fight 2 8](#_Toc96917901)

[2.3.2. Prizefighters 2 9](#_Toc96917902)

[2.3.3. Game Programming Patterns 10](#_Toc96917903)

[2.3.4. Libraries I Will Use 11](#_Toc96917904)

[2.4. Prototyping 12](#_Toc96917905)

[2.5. Modelling 14](#_Toc96917906)

[2.5.1. User Interface 14](#_Toc96917907)

[2.6. Objectives 15](#_Toc96917908)

[2.6.1. A-level Standard Objectives 15](#_Toc96917909)

[2.6.2. Extension Objectives 16](#_Toc96917910)

[2.7. Execution 18](#_Toc96917911)

[3. Documented Design 19](#_Toc96917912)

[3.1. Class Diagram 19](#_Toc96917913)

[3.2. Overview Flowchart 20](#_Toc96917914)

[3.3. Entity-component-system Pattern 21](#_Toc96917915)

[3.4. Entities 22](#_Toc96917916)

[3.5. Components 23](#_Toc96917917)

[3.6. Systems 24](#_Toc96917918)

[3.7. Launch 25](#_Toc96917919)

[3.8. Rendering 25](#_Toc96917920)

[3.9. User Interface 27](#_Toc96917921)

[3.10. Controls 29](#_Toc96917922)

[3.11. Mapping Controls Input to Player Actions 32](#_Toc96917923)

[3.12. Actions 32](#_Toc96917924)

[3.13. Animation 35](#_Toc96917925)

[3.14. Enemy AI 36](#_Toc96917926)

[4. Technical Solution 36](#_Toc96917927)

[4.1. Table of Contents 36](#_Toc96917928)

[4.1.1. Android Module 36](#_Toc96917929)

[4.1.2. Lwjgl3 (Desktop) Module 36](#_Toc96917930)

[4.1.3. Core Module 37](#_Toc96917931)

[4.2. Groups 39](#_Toc96917932)

[4.2.1. Group A 39](#_Toc96917933)

[4.2.2. Group B 39](#_Toc96917934)

[4.2.3. Group C 40](#_Toc96917935)

[5. Testing 41](#_Toc96917936)

[5.1. Table of Tests 41](#_Toc96917937)

[5.2. Testing Video 45](#_Toc96917938)

[6. Evaluation 48](#_Toc96917939)

[6.1. Adherence to Requirements 48](#_Toc96917940)

[6.2. Challenges 48](#_Toc96917941)

[6.3. Possible Improvements 48](#_Toc96917942)

[6.4. Independent Feedback 49](#_Toc96917943)

[7. Code 50](#_Toc96917944)

[7.1. android: top-level 50](#_Toc96917945)

[7.1.1. AndroidLauncher 50](#_Toc96917946)

[7.1.2. TouchControlsGestureListener 51](#_Toc96917947)

[7.2. lwjgl3: top-level 54](#_Toc96917948)

[7.2.1. KeyboardControlsInputProcessor 54](#_Toc96917949)

[7.2.2. Lwjgl3Launcher 57](#_Toc96917950)

[7.3. core: asset 58](#_Toc96917951)

[7.3.1. ComboKingFonts 58](#_Toc96917952)

[7.3.2. ComboKingMusic 59](#_Toc96917953)

[7.3.3. ComboKingSounds 60](#_Toc96917954)

[7.3.4. ComboKingTextures 61](#_Toc96917955)

[7.4. core: comp 63](#_Toc96917956)

[7.4.1. ActionComp 63](#_Toc96917957)

[7.4.2. AIComp 65](#_Toc96917958)

[7.4.3. AnimationComp 66](#_Toc96917959)

[7.4.4. ButtonComp 67](#_Toc96917960)

[7.4.5. GraphicsComp 68](#_Toc96917961)

[7.4.6. HitboxComp 69](#_Toc96917962)

[7.4.7. HPComp 70](#_Toc96917963)

[7.4.8. InfoComp 71](#_Toc96917964)

[7.4.9. ScoreComp 72](#_Toc96917965)

[7.4.10. SliderComp 73](#_Toc96917966)

[7.4.11. TextComp 74](#_Toc96917967)

[7.4.12. TransformComp 75](#_Toc96917968)

[7.5. core: event 77](#_Toc96917969)

[7.5.1. GameEvent 77](#_Toc96917970)

[7.5.2. GameEventManager 78](#_Toc96917971)

[7.6. core: screen 80](#_Toc96917972)

[7.6.1. ComboKingScreen 80](#_Toc96917973)

[7.6.2. GameOverScreen 81](#_Toc96917974)

[7.6.3. GameScreen 84](#_Toc96917975)

[7.6.4. MenuScreen 87](#_Toc96917976)

[7.7. core: sys 92](#_Toc96917977)

[7.7.1. ActionSys 92](#_Toc96917978)

[7.7.2. AISys 96](#_Toc96917979)

[7.7.3. AnimationSys 97](#_Toc96917980)

[7.7.4. BackgroundSys 99](#_Toc96917981)

[7.7.5. CameraSys 102](#_Toc96917982)

[7.7.6. DebugRenderSys 103](#_Toc96917983)

[7.7.7. GameOverSys 104](#_Toc96917984)

[7.7.8. HPRenderSys 105](#_Toc96917985)

[7.7.9. KillingSys 106](#_Toc96917986)

[7.7.10. MusicSys 107](#_Toc96917987)

[7.7.11. PlayerInputSys 108](#_Toc96917988)

[7.7.12. ScoreSys 111](#_Toc96917989)

[7.7.13. SpawningSys 112](#_Toc96917990)

[7.7.14. SpriteRenderSys 117](#_Toc96917991)

[7.7.15. TextRenderSys 119](#_Toc96917992)

[7.7.16. TimeSys 121](#_Toc96917993)

[7.8. core: util 122](#_Toc96917994)

[7.8.1. Util 122](#_Toc96917995)

[7.9. core: top-level 124](#_Toc96917996)

[7.9.1. CKLogger 124](#_Toc96917997)

[7.9.2. CKPrefs 125](#_Toc96917998)

[7.9.3. ComboKing 126](#_Toc96917999)

[7.9.4. UIInputProcessor 128](#_Toc96918000)

[8. Appendix 131](#_Toc96918001)

[8.1. Initial Communication with Third Party 131](#_Toc96918002)

[8.2. Feedback from Third Party 132](#_Toc96918003)

[9. References 133](#_Toc96918004)

# Analysis

## Description

### Gameplay

I will be developing an endless running and fighting game that will run on Android, Windows, MacOS and Linux. Windows, MacOS and Linux will all use Lightweight Java Game Library, (LWJGL). In my report I use the terms desktop and PC interchangeably to refer to these three platforms. Essentially, the player will be running and jumping between the ground and a raised platform and tackling various enemies. It will be 2D from a side camera perspective (like in most platformer games such as Super Mario Bros; see the images in the Prototyping section). But unlike Super Mario Bros, there won't be an end point. The objective is to make it as far as possible before losing all your health points.



There will be various different types of enemy, each with noticeably different behaviour. For example, there could be one type which has lots of health and tries to block your path, and there could be another type which attacks in large groups, and they coordinate to surround you.

As for how the combat will work, it will be mostly melee. This is because I can't see how I can make ranged combat fun – there isn't much variety in shooting from afar. On the other hand, there can be lots of variety in melee combat: high punches, low punches, spinning punches, uppercuts, low kicks, front kicks, high kicks, turning kicks and jumping kicks. My idea is to have all of these different moves available to the player, and each has their advantages and disadvantages.

Another important aspect of the gameplay will be moving between the two levels (the ground level and the raised platform), to stay in a location that makes it easier to combat. For instance, the player may want to stay on the upper level as much as possible, so that they can eliminate enemies easily by jumping down and performing a drop-kick. But if enemies come at you from the upper level, you may want to go down to the lower level so that you can run past them.

There will be several methods of switching between the 2 levels. First, there will be the self-explanatory staircases and ladders, which appear from time to time, and offer an easy way to go up or down. But you should also be able to jump up on top of a table on the ground level and jump up, grabbing hold of the banister and climbing over it onto the first floor. Furthermore, if you are on the upper level and wish to descend, you should be able to climb over the banister, and drop to the floor, and roll. If you fail to time the roll correctly, you will receive fall damage. These last two methods will be difficult to pull off (especially to pull off quickly), so they will be an alternative method aimed at experienced players, and are intended to add more variety to the game.

Above is my initial description of the game. Since writing the rest of the analysis, I have decided to add some features and I've decided in more detail how the game will work. The description above is still correct, it's just missing those extra features. See my objectives for a more comprehensive description of my game.

### Controls

My initial idea was to have the player's movement fully automatic, controlled by an intelligent algorithm, however now I think that moving around is part of the fun and the strategy. I will still have to write that intelligent algorithm to control the movement of the enemies.

On Android, the game will use gesture controls (i.e. tapping and swiping different areas of the screen); I think this will create a tactile and satisfying experience for the player. I'm not entirely decided on what the controls will be yet, but perhaps the best approach would be that the left thumb is used for moving around (using 4 virtual, on-screen buttons), and the right thumb is used for attacking/blocking. Then, you could punch by swiping the right-hand side of the screen with your right thumb. Maybe you could swipe diagonally upwards for a high punch, or directionally downwards for a low punch.

Of course, on the desktop, the controls will have to be different. My initial idea is that the left hand will control movement using the W, A, S and D keys. The A and D keys could be used to run left or right respectively, the S key to roll or go down to the ground level, and the W key to jump or go up to the platform. The right hand will be used for attacking. Maybe it could simply be using the mouse to perform the same gestures as on Android. In other words, to punch you would click and drag the cursor to the right.

One concern I have with these controls is that they require more work from the user. Usually in a mobile fighting game, all the user has to do to punch or kick is tap a button. I want to try something new for my game as I think it would be more satisfying to attack using a combination of different swipes than simply pressing buttons, but I might be wrong – it might be too difficult to learn. I will decide what the actual controls will be through prototyping and testing to see if it feels natural and is comfortable for the user.

### Theme

The protagonist of the game (the character that the user controls) will be a woman called Jane Pond. She is a spy who was working undercover in a foreign intelligence headquarters, but she got found out, so she has to run for her life. However, she wasn't carrying a weapon (so that she didn't attract attention), so she must fend for herself empty-handed.

This second paragraph was added after coding the project. To save time, rather than draw all of the animation frames for this fictional Jane Pond character, I used pixelated images of myself.

## End User

The game will be targeted at teenagers to young adults, because that is the age group which plays the most live-action computer games. My third party will be Rudy Moran, a year 13 sixth form student.

Rudy has experience playing PC and Android games, so his opinion is valuable help. I did prepare some questions for him, but the conversation diverged, and the result was more useful than the questions anyway.

There are many features that were mentioned in the conversation that I could add to my game, however I must be careful not to add too many features because that would make it too hard to code and maintain. I have outlined beside each of my notes my thoughts on it, and the extent to which I will implement it in my game.

My notes for the conversation are in Appendix 1.

I also asked Rudy for his opinion on using touch gestures on Android rather than virtual buttons, which most mobile games use. He liked the idea.

## Research

### Shadow Fight 2



Shadow Fight 2[3] is a mobile game in which the player combats artificial opponents in a 1 versus 1 battle. The strategy lies in timing your moves so that you hit your opponent when they are most vulnerable, for example when they are in the middle of throwing a kick themselves. You also have to learn the correct distance from your opponent from which to attack; otherwise your attacks miss. For instance, a kick must be done at a greater distance than a punch. I can incorporate similar behaviour in my own game.

There is much to learn here that I can apply to my project. One is that I should exaggerate the length of time it takes to perform these attacks, to give the opponent (whether it's the player, or the AI) time to be able to intercept it. If the attacks are too fast, there is no time to react. However, I could include enemies that have slightly faster attacks and are therefore more difficult to face.

One thing I didn't like about this game is that as far as I can tell, although different opponents look different and use different weapons, they behave in largely the same way. This made the game repetitive as it felt like I was doing the same thing over and over. In my game, I will make sure that the different enemies behave significantly differently. Furthermore, in Shadow Fight, the battles always took place in a flat, featureless area and were always 1-on-1, which contributed to this monotonous feeling. In my game, there will be different platforms and multiple enemies coming from different directions, so no two fights will be the same.

Finally, one more thing the game does well is how it seamlessly transitions from move to move. From what I can see, the game does this by always bringing the player back to the same rest position after each move. That way, for each move, you just have to animate it being performed from rest, to the move, and then back to rest. This saves on having extra animations for going directly from each move to each other move. Moreover, if you press the buttons to perform another move when your character is just about to finish a move, it will wait until the current move finishes and then perform the new move. But this only applies to near the end of a move: if you attempt to do another move immediately after *starting* a move, your input will get ignored. To decide how I want this to work in my game, I will write different prototypes for each method when I do my technical solution, so my third party and I can decide which one feels the best.

### Prizefighters 2



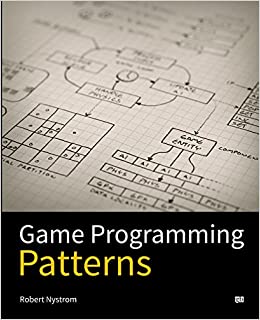
Prizefighters 2[4] is also a mobile game. In this game, the player has a 1v1 boxing fight against a computer-controlled opponent. The game essentially has two parts: the career aspect, where you manage your boxer's career and see him or her progress; and the actual fighting aspect, i.e. learning how to combat different opponents and exploit their weaknesses. I do not plan to have a similar career aspect to my game, so I will focus on the fighting instead.

As you can see from the image, the perspective is pseudo-3D, with the player's character appearing in front of the computer-controlled opponent. However, it still uses 2-dimensional assets, and the camera's perspective is fixed. This is in contrast to Shadow Fight 2, where the characters appear side by side. As my game is an endless runner, the character will be moving through a world, so to show that the character is moving I would have to have a backdrop moving past the character (like in Temple Run). This would mean 3D assets, which would add too much unnecessary complexity to my game. I want my game's technical complexity to be in other areas, such as the enemies' AI, therefore I will use a camera perspective similar to Shadow Fight and not Prizefighters.

After playing the game for around one hour, I found the gameplay fairly simple and repetitive. It seemed to consist of guarding yourself until your opponent stops throwing punches, then throwing some punches back. There is a gameplay feature that aims to make it more interesting: *special punches*. The in-game tutorial attempts to explain how they work, but when I tried to use them, they didn't seem to deal much damage, and I was better off just focusing on the easy but boring blocking and punching.

I was concerned I might give Prizefighters an unfair analysis due to not spending much time learning how to play it; I had played Shadow Fight for 2 years for my own enjoyment. For this reason, I did some extra research by reading an article by Tim at levelwinner.com.[1] This did give me useful tips, such as, "You can also stun your opponent by landing a punch right before they launch one." However, I feel that a game should be fun before you've mastered it; you shouldn't have to do research or spend many hours to enjoy the game. I will try to make my game adhere to Bushnell's law of game design[2], "easy to learn and difficult to master", although this is very difficult, so I may not be able to accomplish it.

### Game Programming Patterns



Game Programming Patterns[5] by Robert Nystrom is a book that describes the benefits and drawbacks of various programming patterns that can be used in games. The code examples used in the book are in C++, but as Nystrom states, "that isn’t to imply that these patterns are only useful in that language"[7]. The actual patterns are language-agnostic. Here I will discuss some tips and patterns I got from this book.

The first tip is that "the measure of a design is how easily it accommodates changes"[6]. Making sure the code is easy to change is the whole point of good architecture. He elaborates, saying that the first step of adding a new feature is to learn the existing code that you are working with, and that that is often the most time-consuming part. He recommends decoupling the code, which means that a new programmer would only have to learn about the specific section of the program they are working on.

Nystrom also discusses the downsides of flexibility, one of which is performance. He says "performance is all about assumptions"[8] and making a system more flexible reduces the assumptions we can make. This is something I have struggled with in the past with making my own games; I am sometimes uncertain whether to make my program open to big changes or to limit it, so I can make it faster. I think that for my NEA project, performance will not be of much concern, as my game won't be particularly complex or CPU-intensive. So I will focus on making my game loosely coupled and easy to be changed.

Of the patterns, the one I found most applicable to my game was the Service Locator pattern. In this pattern, code that performs a task that may be required by various parts of the game (for example, playing audio) is called a *service*. Each service has an interface without any implementation, and one or more *service providers*, which actually perform the task. Before the service needs to be used, external code registers one of the service providers with a service locator. Then, when you want to use the service, you call a method in the service locator, which returns the service provider that was registered.

The reason I believe this would be very helpful for my game is that I could have separate service providers for Android and PC, if I need to. While libGDX allows me to play audio on Android and PC using the same API, I will need two different controls systems because PC will use the keyboard whereas Android will use touch gestures. If I used the service locator pattern, I could just register the service providers in the small, platform-specific module and then the service locator would take care of the rest. I could also have additional service providers that aid in debugging the code, such as a *NullAudioService* (which implements *AbstractAudioService* but doesn't do anything), or a *LoggedAudioService* (which behaves the same as *AudioService* but also logs a message).

### Libraries I Will Use

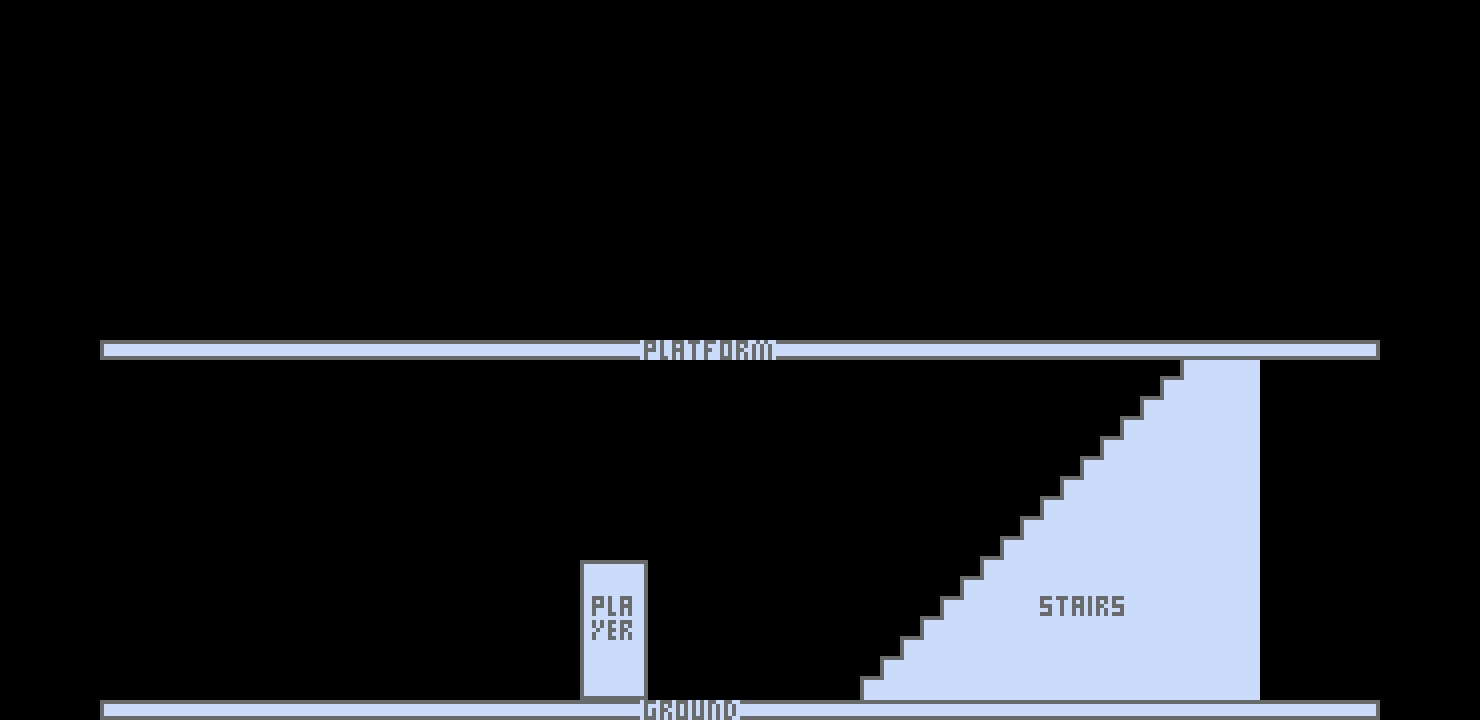
To develop my game I will use libGDX[9], which is a cross-platform Java game development framework. It provides the basics for what I need in my game (such as audio, graphics, user input and maths APIs), while still being flexible – it doesn't tie you into a specific approach, like Unity does. Also, there are other frameworks that are made by the libGDX project but are optional extensions. One of these that I will use is Ashley, which is a tiny framework for the entity-component-system pattern. I will discuss ECS in the Modelling section.

However, the language I will be using is Kotlin[10], not Java, primarily because I am more proficient at it. In addition, its main benefits over Java are conciseness and null-safety. As it is completely interoperable with Java, I see no downsides to using it. In fact, there is already a library called libKTX[11] which adapts libGDX to better take advantage of Kotlin's benefits. I will also use libKTX in my project.

For any physics, I will likely use Box2D[12] as it works well with libGDX (they have created a Java/libGDX wrapper for it) and provides all that I need. I may not need much physics in my game, as it is mostly jumping on platforms, but using Box2D will give me the ability to add nearly anything I want to my game without worrying about its physics being too difficult for me. I want the other areas of my game to be the complex parts, code that is specific to my game and that there are no pre-existing libraries for.

Finally, I may also use Scene2D[13] for the user interface. Scene2D is a part of libGDX designed for managing text, buttons, menus and other UI elements. It expects you to provide any graphical assets in the form of its Skin class and JSON files that reference images. However, the libGDX tests provide some assets that anyone is allowed to use, so I will use them.

## Prototyping



Before I can even get started on the game, there is a lot of low-level coding that needs to take place, to set up all the libraries I am using and do all sorts of things that aren't directly related to the game. It didn't take too long though, as I could copy and adapt my code from my other libGDX project.

The project is divided into 3 modules: android, core and lwjgl3. android contains code specific to the Android version; lwjgl3 is code for the Windows/Linux version (libGDX uses a library called LWJGL behind the scenes); and core is for all other code that isn't specific to a platform. core will be the largest module by far as it will contain pretty much the whole game. android and lwjgl3 will contain just the launcher code.

Now I will briefly explain how my prototype works, on a high level, using the most important core code. See Entity-component-system Pattern in the Documented Design section for a detailed explanation of entities, components and systems.

// add entities

val player = engine.entity {

with<TransformComponent> {

setSizeFromTexture(textures.prototype\_player)

rect.setPosition(120f, 5f)

}

with<GraphicsComponent> {

sprite.setRegion(textures.prototype\_player)

}

with<PlayerComponent> {}

}

engine.entity {

with<TransformComponent> {

setSizeFromTexture(textures.prototype\_ground)

rect.setPosition(0f, 0f)

}

with<GraphicsComponent> {

sprite.setRegion(textures.prototype\_ground)

}

}

engine.entity {

with<TransformComponent> {

setSizeFromTexture(textures.prototype\_platform)

rect.setPosition(0f, 90f)

}

with<GraphicsComponent> {

sprite.setRegion(textures.prototype\_platform)

}

}

engine.entity {

with<TransformComponent> {

setSizeFromTexture(textures.prototype\_stairs)

rect.setPosition(190f, 5f)

}

with<GraphicsComponent> {

sprite.setRegion(textures.prototype\_stairs)

}

}

//add systems to engine

engine.run {

addSystem(RenderSystem(batch, gameViewport))

}

First, I create my entities and add them to the engine. This occurs in GameScreen.show. For each entity I initialise the components that the entity will have. For example, the ground entity has a TransformComponent and a GraphicsComponent. I set the size of the TransformComponent to the dimensions of the ground sprite, and I set the position to the origin. Then I set the texture of the GraphicsComponent to the ground sprite.

override fun update(deltaTime: Float) {

gameViewport.apply()

batch.use(gameViewport.camera.combined) {

super.update(deltaTime)

}

}

override fun processEntity(entity: Entity, deltaTime: Float) {

val transformComp = entity.getNotNull(TransformComponent.mapper)

val graphicsComp = entity.getNotNull(GraphicsComponent.mapper)

if (graphicsComp.sprite.texture == null) {

log.error { "Entity $entity has no texture for rendering" }

return

}

if (!graphicsComp.visible) return

graphicsComp.sprite.setBounds(transformComp.rect)

graphicsComp.sprite.rotation = transformComp.rotation

graphicsComp.sprite.draw(batch)

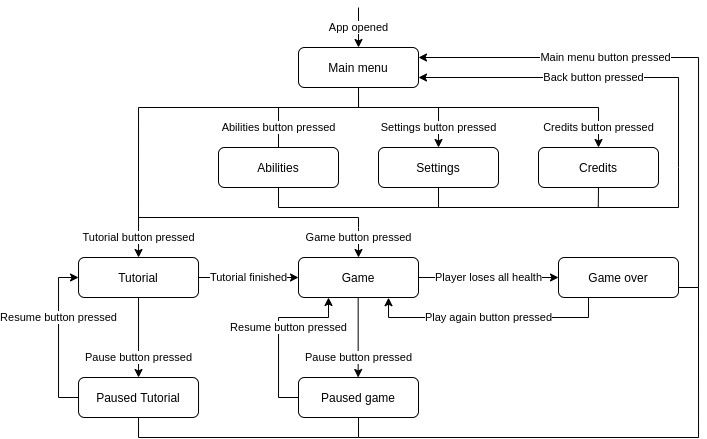
}

Then, I add all the systems to the engine. My prototype only uses one system: the RenderSystem. This system iterates over all entities that have both a TransformComponent and a GraphicsComponent, and draws them onto the screen. I override the update method, which is called once per frame and would usually iterate over all the entities and call processEntity for each one with both a TransformComponent and a GraphicsComponent. I change it to first apply the gameViewport, which moves the camera so that any subsequent rendering is in world coordinates. Next I use the SpriteBatch, which means that all the render calls inside it are just drawing onto the batch, and then at the end of the block the batch will optimise them for processing my the GPU and then send them off.

In processEntity (this gets called once for each entity that should be rendered), I get the entity's TransformComponent and GraphicsComponent. I then ensure that the GraphicsComponent has a texture and that it is set to be visible. Finally, I set the bounds and rotation of the sprite as specified by TransformComponent, and I draw it.

## Modelling

### User Interface



When the user opens the app, they will see Jane Pond spying undercover on the left (an animated loop). On the right there is a large screen displaying the highscore and 4 buttons: "Play", "Abilities", "Settings" and "Credits".

If the user taps the "Settings" button, various settings will be displayed on the screen (e.g. SFX volume, music volume, show tutorial) along with a button to go back.

The "Credits" button will display the credits along with a button to go back.

If you press the "Abilities" button, the screen will display the amount of XP the user currently has, and the current level of each of Jane's abilities, each with a button to upgrade it. For example, there will be a max health ability. The screen will display the current max health, with a button to upgrade it. The button should display the cost of upgrading and the amount the max health would go up by should you press the button. There should also be a button to return to the main menu.

If you press the "Play" button, a siren will start, and a picture of Jane Pond will appear on the screen, along with the flashing words, "IMPOSTOR DETECTED". Jane will start running and this will transition smoothly into the game.

If the user hasn't completed the tutorial since installing the app, or the "Show tutorial" option was enabled in the settings, the "Play" button will be replaced by a "Tutorial" button. When this is pressed, the game will start in tutorial mode. This means that it interactively shows you how to play the game. At the end of the tutorial, it transitions smoothly into the real game.

The game UI will consist of the score displayed in the top left, along with a pause button. When the pause button is pressed, the game logic and animation is suspended, and a popup appears stating that the game has been paused. This popup contains a button to resume the game.

Finally, when the user loses all of their health, a popup appears saying that you have died. It also states your final score and highscore and has buttons to play again and to return to the main menu.

## Objectives

Since writing the rest of my analysis, my idea for the game has changed somewhat. This list of objectives is the most complete and up-to-date definition of what my game should be like. In addition, I have deliberately left out details on exactly how the controls for movement and combat will work, as this is likely to change many times and I will need to decide this through rapidly iterating between prototyping and testing and asking my third party for his opinion.

### A-level Standard Objectives

1. Main menu
   1. The "play" button should start the game
   2. The "settings" button should display the settings
   3. The "credits" button should display the credits
   4. In the main menu, to the right of Jane, there should be a display screen (i.e. in the intelligence headquarters, Jane is near a large screen. It is imagined that there are many of these screens throughout the building which are used to communicate messages to everyone)
   5. The buttons should be displayed on this screen
   6. The highscore should also be displayed on this screen
2. Settings
   1. There should be a setting to change SFX volume
   2. There should be a setting to change music volume
   3. There should be a button to return to the main menu
3. Credits
   1. All the credits should be displayed
   2. The user should be able to scroll (unless all the credits fit on the screen)
   3. There should be a button to return to the main menu
4. Game UI
   1. There should be the current score displayed at the top
   2. The current health should also be displayed
   3. There should be a pause button displayed
   4. Pressing the pause button pauses the game logic and animation
   5. Pressing the pause button displays a popup
   6. The popup indicates that the game is paused
   7. The popup includes a button to resume the game
   8. The popup includes a button to return to the main menu
   9. The resume button resumes game logic and animation and closes the popup
   10. The main menu button discards the user's progress in the game and returns to the main menu, without saving the score
5. Movement and physics
   1. Jane should be able to run left and right
   2. Jane should be able to jump
   3. Jane should be able to go up or down between the ground and upper level
   4. Jane should not be able to walk through enemies; they should block her path
6. Combat
   1. Jane should be able to punch enemies
   2. There should be at least 3 different types of punches
   3. Jane should be able to kick enemies
   4. There should be at least 3 different types of kicks
7. Enemies
   1. There should be at least 3 different types of enemy
   2. Each type of enemy should behave noticeably differently
   3. The enemies should be controlled by a rule-based AI
   4. The enemies should appear to behave intelligently
   5. Enemies should be able to deal damage to Jane, which decreases her health
8. Game logic
   1. The game should render a background behind Jane
   2. Staircases, ladders, objects that Jane can climb on top of, enemies and items should appear randomly
   3. Staircases, ladders, objects that Jane can climb on top of, enemies and items should not appear too close together or too far apart, making the game unfairly easy or hard
   4. The difficulty of the game should increase the further the player gets – this should be done by making tough enemies appear more often and making everything good for the player appear less often
   5. The score increases when Jane progresses to the right
   6. The score also increases when Jane eliminates enemies
9. Sound
   1. A music loop should be played in the main menu
   2. A different loop should be played in the game
   3. Sound effects should be played whenever it makes sense (e.g. the player or an enemy was hit)
10. Game over screen
    1. When the player runs out of health, a game over screen should be displayed
    2. The score should be displayed
    3. The highscore should be displayed
    4. There should be a button to play again
    5. There should be a button to return to the main menu
    6. If the user has beaten their highscore, this should be indicated
    7. The highscore should be saved so that it persists when the app is closed

### Extension Objectives

1. Extra-nice main menu screen
   1. Jane could be animated to appear to be spying undercover before she is found out (before the game starts)
   2. When the "play" button is pressed, there could be a short animation where a picture of Jane Pond will appear on the screen, along with the flashing words, "Impostor detected"
   3. Jane could then start running towards the right
   4. Then, the camera could zoom out and the game UI (i.e. the score and pause button) could appear (thereby seamlessly transitioning from the main menu to the game)
2. Abilities screen
   1. There could be a button in the main menu to open the abilities screen
   2. Could display the amount of XP the user currently has
   3. Could display the current level of each of Jane's abilities
   4. Each ability could have a button to upgrade it
   5. Could display the cost of upgrading each ability
   6. Could display the amount the ability level would go up by if the user were to press the button
   7. Pressing a button could increase the ability level by the amount specified and decrease the XP by the amount specified
   8. There could be a button to return to the main menu
   9. The level of each of Jane's abilities and the user's current XP level could persist when the app is closed
3. Tutorial
   1. There could be a tutorial that teaches you the controls for the game
   2. It could also teach you strategies on how to play the game well
   3. It could be interactive, i.e. it teaches you as you play the game
   4. There could be different levels of tutorial: a basic tutorial to get started, an intermediate tutorial, and an advanced tutorial
   5. There could be a button in the main menu that starts the tutorial
4. More movement options
   1. Jane could be able to roll
   2. Jane could be able to go up or down staircases
   3. Jane could be able to climb on top of large objects
   4. Jane could be able to jump from large objects to climb onto the upper level
   5. Jane could be able to drop from the upper level down to the ground level
   6. Jane could only be able to jump when she is on solid ground
   7. When Jane drops from a large height (e.g. from the upper level or from near the top of a ladder or staircase), if the user does not do a forward roll within 500 ms of touching the ground, Jane's health is decreased
5. Background objects
   1. There could be objects that appear randomly, such as a desk, a plant or a door
   2. Jane would not be able to interact with these objects; she would walk past them as it they weren't there
6. Items
   1. There could be at least 3 different types of item, including an ammunition item and a bandage item
   2. When Jane touches an item, the item's effect could be applied
   3. When touching the ammo item, Jane's pistol is filled back up to its maximum capacity
   4. The ammunition item could be rare enough so that the player is usually forced to use punch and kick rather than use the pistol
   5. When touching the bandage item, Jane's health is increased (but not past the maximum health)
   6. Jane could be able to shoot enemies with her pistol
   7. Jane could only be able to shoot when she has at least one round in her pistol
   8. There could be other weapons that Jane could pick up and use, such as a knife
7. Missions
   1. There could be various missions available to complete
   2. They could be displayed on groups of 3
   3. When the user completes all 3 missions, their level could increase and they could be presented with the next 3 missions
   4. The user's current level could be displayed on the main menu
   5. The 3 active missions could be displayed in the main menu
   6. The 3 active missions could be displayed for a few seconds after the user presses the "play" button
   7. The missions that are already completed could be highlighted to indicate that they're completed
8. Coins
   1. Coins could appear in the game
   2. The player could collect the coins when they run through them
   3. The current number of coins that the user has could be displayed in the shop (see below)
   4. The game over popup could display the number of coins gained in that run
9. Shop
   1. There could be another button in the main menu to open the shop screen
   2. In the shop screen, the player's current number of coins could be displayed
   3. The user could buy various different items using their coins
   4. Each item could cost a specific number of coins
   5. Some items could be consumable (meaning they are used up in the game and to use them again the user would have to buy them again)
   6. The user could be able to buy more than one of each consumable item
   7. Some items could be permanent (meaning they are not used up in the game and once purchased they can be used forever)
   8. The user shouldn't be able to buy more than one of each permanent item

## Execution

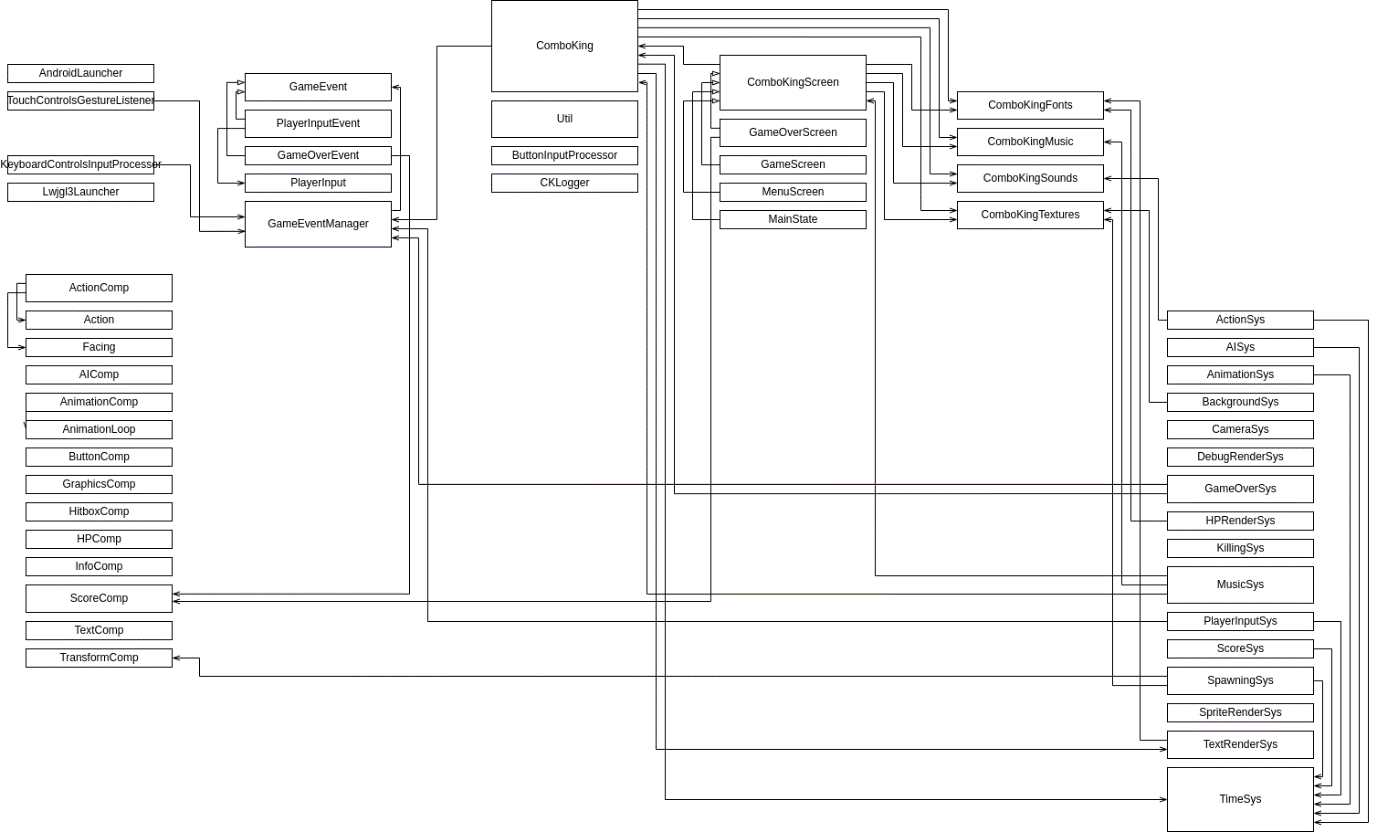
The critical path for my game (the crucial steps that are most important to get done) are:

1. Main menu with a "play" button
2. Score displayed at the top of the screen
3. Jane can run left and right
4. Score increases as Jane runs to the right
5. Enemies appear randomly
6. Jane can attack the enemies
7. The enemies can decrease Jane's health
8. When Jane's health reaches zero, the game is over
9. The game over popup displays the final score
10. The game over popup has a button to play again

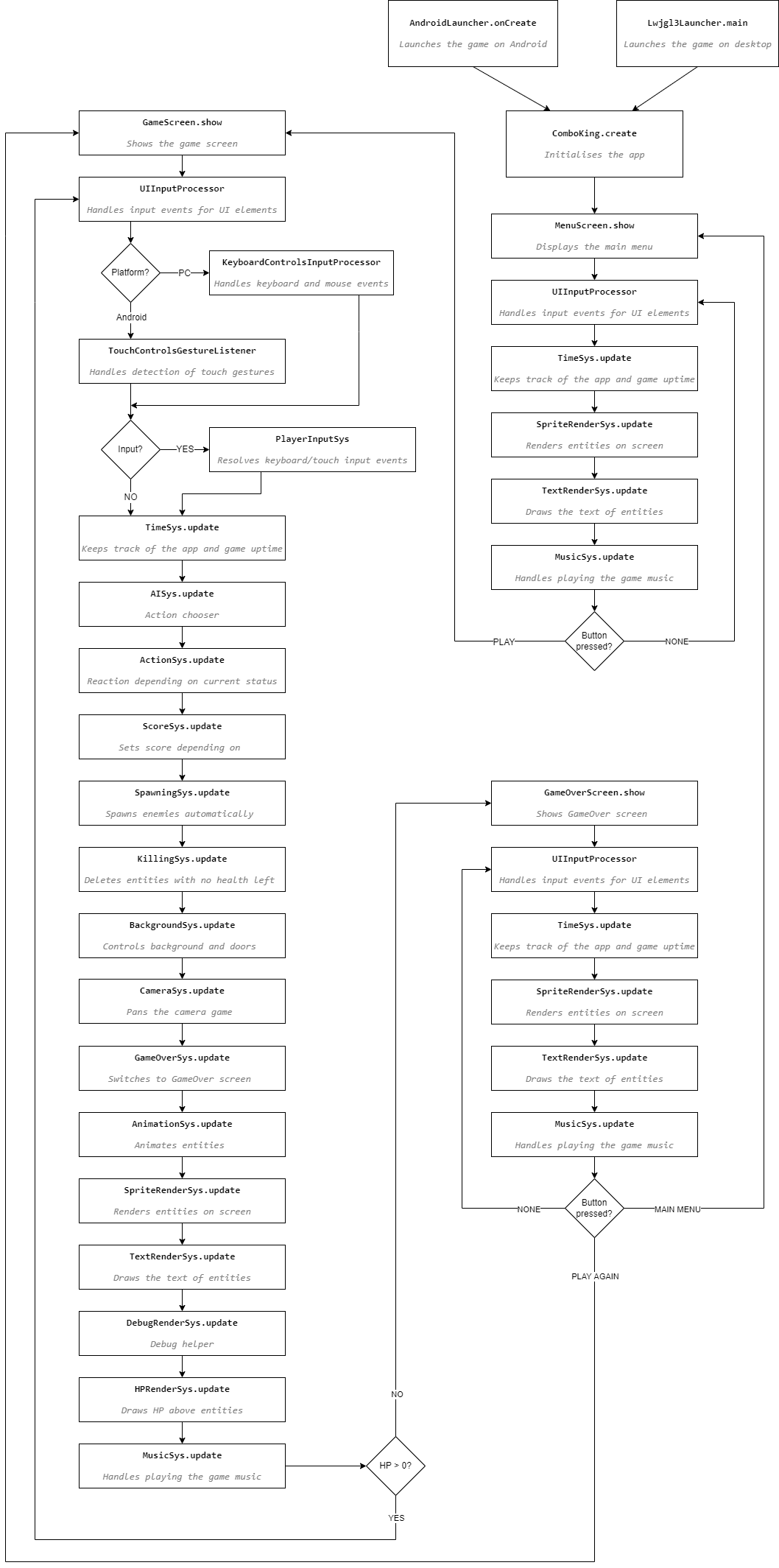
# Documented Design

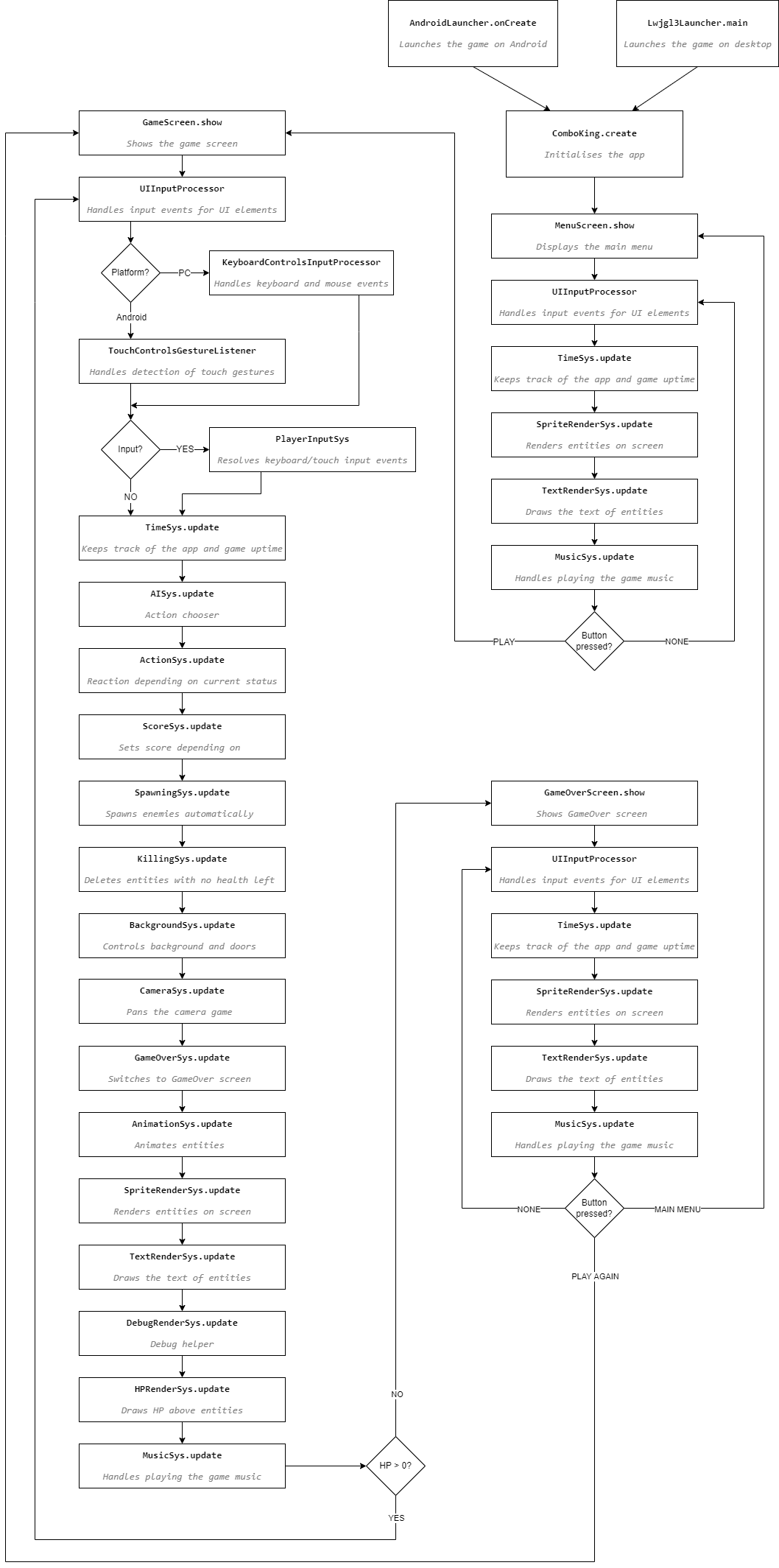
## Class Diagram

Below is a class diagram of the project. Directed associations (Directed association) signify that the first class has a member variable of the second class's type. Generalisation associations (Generalisation association) signify that the first class is a subclass of the second class.



## Overview Flowchart





## Entity-component-system Pattern

In my project I used the entity-component-system architectural pattern, which I believe to be a very good pattern for separating logic and data, increasing cohesion and reducing coupling. In addition, ECS is primarily composition-based rather than inheritance-based, which makes it more flexible and easier to maintain and adapt. The entities, components and systems make up the majority of the code. See the technical solution section table of contents for a brief description of every component and system.

## Entities

* Just a bag of components
* Often initialises each component with specific values, e.g. a player entity might specify the coordinates of the transform component
* Examples are: the player, a specific enemy, a specific item, a specific flight of stairs, and perhaps also abstract things such as game state
* They can be hard-coded or generated dynamically
* In libGDX they are represented by an object
* You can add and remove them from the engine to remove them from the game, while their state is still saved in the object
* I initialise some entities from a screen's show method (e.g. player) and some entities dynamically from a system (e.g. the enemies)

Everything that should be visible on the screen or should store data is represented by an entity. These entities are added to the engine either by a screen's show method, or dynamically by a system. For example, the "PLAY" button is defined like this:

engine.entity {

with<InfoComp> {

name = "PlayButton"

}

with<TransformComp> {

x = 89f

y = 47f

setSizeFromTexture(textures.btn\_play)

}

with<GraphicsComp> {

textureRegion = textures.btn\_play

}

with<ButtonComp> {

onPress = { game.setScreen<GameScreen>() }

}

}

This is executed when the main menu is shown, to add the button the engine. Behind the scenes, first this creates a new entity object. Then, for each component listed, if there is a component of the same type available in the component pool (a pool of components that are no longer in use), it reuses that component. Otherwise, it creates a new one. For each component it also runs its reset method and then executes the configuration that you can see inside the with function. Finally, it adds the new entity to the engine.

First, I configure an InfoComp. I do this for all entities, because it's useful to give them a name so that you can easily see which entity is which when you are debugging.

Inside TransformComp I set the x and y coordinates and also the width and the height. Since the width and height are the same as the dimensions of the texture for most entities, I usually use my helper method setSizeFromTexture, however it's also possible to set the width and height manually.

All I need to do inside GraphicsComp is set the texture that it uses.

Lastly, inside ButtonComp I define what should happen when the button is pressed.

## Components

* Only holds data related to a specific thing
* Because each entity can have any combination of components, ECS is a more flexible approach to reusing code than inheritance
* In libGDX they are represented by a class
* I put all my components in the comp package
* When I add components to an entity, libGDX tries to reuse component instances from deleted entities, or otherwise creates a new instance
* Whenever it reuses an old component, libGDX first calls the reset method to prepare the component to be used again
* Example components: TransformComp (specifies position and orientation of an entity), GraphicsComp (specifies texture and whether it should be rendered)

Each component defines a type of entity, but the types are composition-based instead of inheritance-based. This means that entities can mix and match types, resulting in more flexibility. For example, the TransformComp is defined like this:

class GraphicsComp : Component, Pool.Poolable {

companion object {

val mapper = mapperFor<GraphicsComp>()

}

var textureRegion: TextureRegion? = null

var visible = true

var flippedHorizontally = false

override fun reset() {

textureRegion = null

visible = true

flippedHorizontally = false

}

}

Companion objects are Kotlin's way of defining static variables and methods. So the mapper variable belongs to the class itself, not the instances of the class. The mapper is used for retrieving components from entities efficiently.

Below that you can see that all the component does is define some variables to store the component's data, and define a method which resets the variables to their default state. As the components get reused, all the variables must be mutable.

## Systems

* A system is a class that contains code for a specific part of the game logic
* Systems act on the data stored in different entities' components
* Separating game code into different systems increases cohesion and makes specific sections of code easier to find
* It makes it easier to turn on or off specific functionality
* For example ActionSys, PlayerInputSys, RenderSys
* Systems can also be added and removed from the engine at runtime to turn them on or off
* IteratingSystems are a type of system that iterate over a group of entities that have specific components and perform some task for each entity
* For example, RenderSys is an IteratingSystem that iterates over all entities with a TransformComp and a GraphicsComp and renders each one onto the screen

Here is the code for AISys:

class AISys(

private val player: Entity,

private val timeSys: TimeSys

): IteratingSystem(

allOf(AIComp::class, ActionComp::class).get(), 10

) {

override fun processEntity(entity: Entity, deltaTime: Float) {

val entityAIComp = entity.getNotNull(AIComp.mapper)

val entityMoveComp = entity.getNotNull(ActionComp.mapper)

entityAIComp.state = entityAIComp.determineState(entity, player)

entityMoveComp.startAction(entityAIComp.determineAction(entity, player, entityAIComp.state), timeSys.appUptime)

entityMoveComp.facing = entityAIComp.determineFacing(entity, player, entityAIComp.state)

}

}

AISys is an IteratingSystem, which means that it performs a certain method (namely processEntity) on each entity that has the right components. In this case the requirements are that each entity must have an AIComp and an ActionComp for it to be processed by AISys. I also define the priority of the system, 10, in this case.

Then, in each frame, the processEntity method is executed for each entity. First I retrieve the AIComp and ActionComp of the entity. Then, because the actual AI procedures are defined within each entity's AIComp, all AISys does is it executes them.

## Launch

The project consists of three Gradle modules: android, core and lwjgl3. The vast majority of the code is platform-agnostic and is located in the core module, which leaves the platform-specific code in the android and lwjgl3 modules. There are two entry points for the app: AndroidLauncher and Lwjgl3Launcher, which launch the app on Android and PC respectively. The launcher classes both create ComboKing, which is the main class for the game. android and lwjgl3 both depend on core, but core does not depend on android or lwjgl3. This means that if I wanted to add support for a third platform (e.g. HTML5), in theory I wouldn't have to edit any of the existing modules. All I would have to do is create a new HTML5 module with a HTML5 launcher class.

My main class takes a single argument as a parameter, which is a function that takes a GameEventManager (and the screen's dimensions) and returns an InputProcessor. GameScreen needs that InputProcessor for gameplay-related input, such as moving the player left and right. The InputProcessor is a class that responds to platform-specific input events by triggering the appropriate game events. Therefore, it needs a reference to the GameEventManager. But the GameEventManager is created inside the main class, so we don't have access to it from the launchers. So, I decided that passing in a function from the launcher and then invoking the function from within GameScreen to obtain the InputProcessor was the most sensible solution to the problem.

When the main class is initialised, it initialises ComboKingFonts, ComboKingMusic, ComboKing-Sounds and ComboKingTextures, which immediately load the assets into memory and store them. It's fine to keep them in memory because the fonts, sounds and textures are all small and the music is streamed (so only the part that's currently playing is actually in memory at a given time). The main class also creates the Engine, a libGDX class which manages the entities, components and systems.

Then it sets libGDX's active InputProcessor to a new InputMultiplexer. An InputMultiplexer is a type of InputProcessor that contains an array of InputProcessors and delegates to them. It's essentially a way of combining multiple InputProcessors into one. The InputMultiplexer is initialised containing only one InputProcessor, the ButtonInputProcessor, for detecting when a button has been pressed. When in the menus, this is the only InputProcessor required, but when in the game, the other InputProcessor (which is responsible for detecting gameplay input) is added to the InputMultiplex-er.

Next, the main class creates the screens and adds some systems to the engine. Those systems remain active until the app is closed because they are required on all screens, however most systems are only required in some screens, so they are added when the screen is shown. Finally, the main class shows the MenuScreen.

## Rendering

Everything that should be rendered onto the screen is represented by an entity which has both a GraphicsComp and a TransformComp. The GraphicsComp stores the texture, and the Transform-Comp stores the position and dimensions. Each entity is rendered by SpriteRenderSys.

class EntityRenderingComparator: Comparator<Entity> {

override fun compare(p0: Entity, p1: Entity): Int {

val p0TransformComp = p0.getNotNull(TransformComp.mapper)

val p1TransformComp = p1.getNotNull(TransformComp.mapper)

return when {

p0TransformComp.z < p1TransformComp.z -> -1

p0TransformComp.z > p1TransformComp.z -> 1

p0TransformComp.y < p1TransformComp.y -> -1

p0TransformComp.y > p1TransformComp.y -> 1

else -> 0

}

}

}

SpriteRenderSys is a SortedIteratingSystem which means it iterates over certain entities in a specific order. The order is determined by EntityRenderingComparator (in the same file). Entity-RenderingComparator sorts entities first by the z coordinate, and then by the y coordinate, which means if two entities have the same z coordinate, the one which is lower on the screen will be drawn on top. Entities that are lower down should be in-front because from the perspective of the camera they look closer.

override fun update(deltaTime: Float) {

viewport.apply()

batch.use(viewport.camera.combined) {

super.update(deltaTime)

}

}

Inside the update method, I call viewport.apply, which updates the camera's projection and view matrices in case the camera has been moved or the app has been resized. Then I call the superclass's update method within the context of a batch, which essentially means that any rendering that happens now will actually get rendered onto the screen all in one go, which significantly increases performance. The superclass's update method is the method that iterates over all the entities and calls processEntity on each of them.

override fun processEntity(entity: Entity, deltaTime: Float) {

val graphicsComp = entity.getNotNull(GraphicsComp.mapper)

if (!graphicsComp.visible) return

if (graphicsComp.textureRegion == null) {

log.error { "Entity $entity is set to be visible but has no texture." }

return

}

val transformComp = entity.getNotNull(TransformComp.mapper)

sprite.setBounds(transformComp.x, transformComp.y, transformComp.width, transformComp.height)

sprite.setRegion(graphicsComp.textureRegion)

sprite.setFlip(graphicsComp.flippedHorizontally, false)

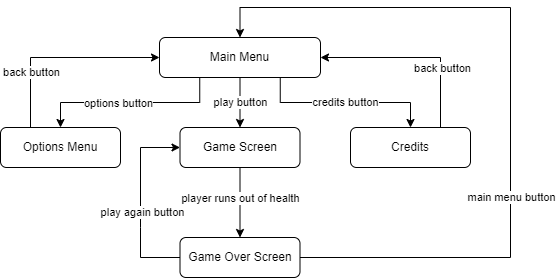
sprite.draw(batch)

}

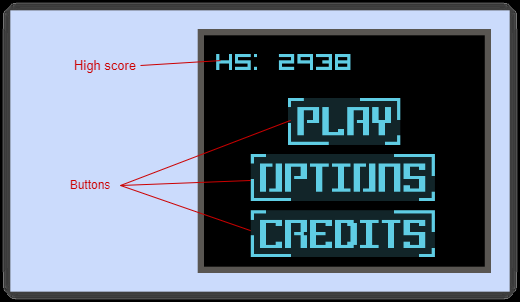
Inside processEntity, I first retrieve the relevant components. I skip the rendering if the entity is set to be invisible or if no texture has been set. Then I make use of a Sprite object (the same Sprite object gets reused each time) to do the rendering, setting the position, dimensions and texture as shown.

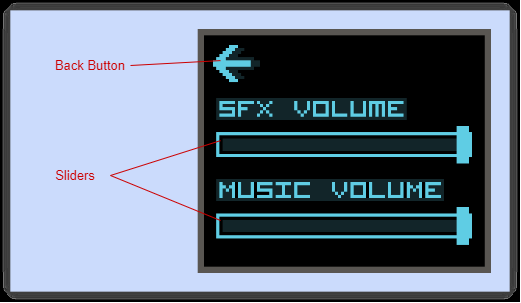
## User Interface

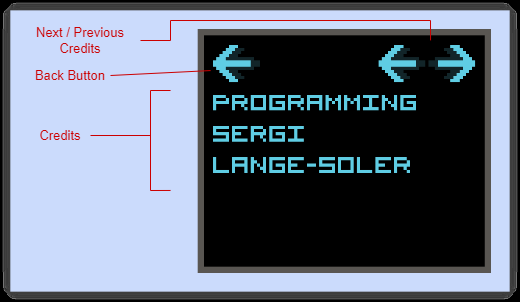
There are 3 different screens in the app, MenuScreen, GameScreen and GameOverScreen. MenuScreen has 3 different states that it can be in. Below is a flowchart describing how to switch between the screens and states.

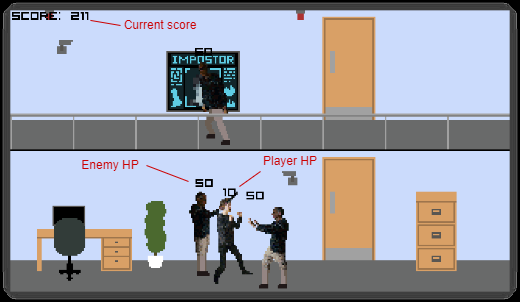


The following diagrams describe the UI elements of each screen.











## Controls

When the GameScreen gets shown, the first thing that happens is I add the player input processor to the input multiplexer. As explained previously, the player input processor that is created depends on the platform that the game is running on – on PC it's KeyboardControlsInputProcessor and on Android it's TouchControlsGestureListener.

//add player controls input processor

playerInputProcessor = createPlayerInputProcessor(gameEventManager, viewport.screenWidth, viewport.screenHeight)

(Gdx.input.inputProcessor as InputMultiplexer).addProcessor(playerInputProcessor)

The keyboard controls are not as simple as you might expect. Whenever a key gets pressed down, I trigger the corresponding input event, for instance:

Input.Keys.A -> {

GameEvent.PlayerInputEvent.input = PlayerInput.LEFT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

return true

}

This has the effect that if two keys are pressed down at the same time, the player will move in the direction corresponding the key that was pressed last. For example, if the user holds down A and then holds down D while still holding down A, the player will run to the left at first and then run to the right as soon as D is pressed. This behaviour is desirable because if the user is quickly switching from moving left to moving right, it is likely that there will be a short period of time when both keys are pressed. If the player only started moving to the right once A is released, the game will feel less responsive.

This functionality has some subtle consequences. Using the previous example of holding down A and then D, if the user then lets go of D, they would expect the player to switch back to running to the left, since A is the only key being pressed. But if A is not being pressed when the user lets go of D, they would expect the player to stop moving. (This works similarly for letting go of the A key.)

Input.Keys.D -> {

if (Gdx.input.isKeyPressed(Input.Keys.A)) {

GameEvent.PlayerInputEvent.input = PlayerInput.LEFT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

else {

GameEvent.PlayerInputEvent.input = PlayerInput.STOP

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

return true

}

Finally, if the player still holding down A or D after pressing W or S to go through a door, they would expect to carry on moving in the same direction when the player comes out of the other door.

Input.Keys.W, Input.Keys.S -> {

if (Gdx.input.isKeyPressed(Input.Keys.A)) {

GameEvent.PlayerInputEvent.input = PlayerInput.LEFT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

else if (Gdx.input.isKeyPressed(Input.Keys.D)){

GameEvent.PlayerInputEvent.input = PlayerInput.RIGHT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

return true

}

On Android, user input is controlled by TouchControlsGestureListener. The controls are as follows. The left side of the screen is used for movement and the right side of the screen is used for combat. I made it such that if the user starts swiping on the left side and accidentally moves over to the right side during the swipe, the whole swipe still counts as movement input, so the player doesn't end up punching. To move left, the user swipes from right to left the left side of the screen and keeps their finger on the screen. To move right, the user swipes from left to right the left side of the screen and keeps their finger on the screen. To stop moving, the user lifts their finger off the screen. To use doors, the user swipes upwards or downwards on the left side of the screen while near a door. To punch to the left, the user swipes from right to left on the right side of the screen. To punch to the right, the user swipes from left to right on the right side of the screen.

I implemented the above as follows. I keep track of the current position of the finger on the screen in touchHoldXMove, touchHoldYMove and touchHoldXCombat. I also keep track of whether the user is using the left side of the screen or the right side of the screen in usingLeftControls and usingRightControls. in touchDown, I store the initial position of the finger and what side of the screen we are using.

if (x < screenWidth/2f) {

touchHoldXMove = x

touchHoldYMove = y

usingLeftControls = true

}

else {

touchHoldXCombat = x

usingRightControls = true

}

Then, inside of pan, if using the left-hand side of the screen, I first calculate the x distance, y distance and y displacement that the finger has travelled from the last stored position. The y displacement is positive if the finger moved upwards and negative if the finger moved downwards. The x and y distances are always positive regardless of direction. Throughout the pan method, I make comparisons between the distance the finger has moved and a constant MIN\_SWIPE\_-DISTANCE. This is so that the game doesn't detect tiny swipes while the user is trying to hold down on the screen in the same place. Then, if distY > distX it means that the swipe moved more in the y direction than in the x direction, so the swipe was either upwards or downwards. So, I trigger the appropriate events and take note of the new finger position. Otherwise, distY <= distX so the swipe was either leftwards or rightwards. Then once again I trigger the appropriate game events and store the new finger position. Finally, if using the right-hand side of the screen, the code is simpler because I only have to discern between swipes to the left and swipes to the right.

//LHS controls

if (usingLeftControls) {

val distX = abs(x - touchHoldXMove)

val dispY = touchHoldYMove - y

val distY = abs(dispY)

//using doors

if (distY > MIN\_SWIPE\_DISTANCE && distY > distX) {

if (dispY < 0) { //swipe down

touchHoldXMove = x

touchHoldYMove = y

GameEvent.PlayerInputEvent.input = PlayerInput.DOWN\_STAIRS

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

else { //swipe up

touchHoldXMove = x

touchHoldYMove = y

GameEvent.PlayerInputEvent.input = PlayerInput.UP\_STAIRS

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

}

else if (x < touchHoldXMove - MIN\_SWIPE\_DISTANCE) {

touchHoldXMove = x

touchHoldYMove = y

GameEvent.PlayerInputEvent.input = PlayerInput.LEFT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

else if (x > touchHoldXMove + MIN\_SWIPE\_DISTANCE) {

touchHoldXMove = x

touchHoldYMove = y

GameEvent.PlayerInputEvent.input = PlayerInput.RIGHT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

}

// RHS controls

else {

if (x < touchHoldXCombat - MIN\_SWIPE\_DISTANCE) {

touchHoldXCombat = x

GameEvent.PlayerInputEvent.input = PlayerInput.PUNCH\_LEFT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

else if (x > touchHoldXCombat + MIN\_SWIPE\_DISTANCE) {

touchHoldXCombat = x

GameEvent.PlayerInputEvent.input = PlayerInput.PUNCH\_RIGHT

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

}

Lastly, when the user lifts their finger, I trigger a stop event if the left-hand side is being used, and I set usingLeftControls and usingRightControls to false.

if (usingLeftControls) {

GameEvent.PlayerInputEvent.input = PlayerInput.STOP

gameEventManager.trigger(GameEvent.PlayerInputEvent)

}

usingLeftControls = false

usingRightControls = false

## Mapping Controls Input to Player Actions

To increase cohesion and flexibility, I implemented the input processors in such a way that they are not coupled to the actions. This means that when the input processor triggers an input event, the input event contains a PlayerInput type (which is represented by an enum) and the PlayerInput types don't map exactly one-to-one with the Action types.

Perhaps this is better explained with an example. When the user swipes from right to left on the right side of the screen, if the player is facing left, they will perform a normal punch, whereas if the player is facing right, they will perform a spinning punch. Therefore, even with the same input, the action that is performed can depend on the game state. So, to avoid coupling the input processors with the game state, I had to use 2 different enums, PlayerInput and Action, which are similar but not quite the same. PlayerInputSys is the system that maps PlayerInputs to Actions.

The eventCallback is the function that gets called when a PlayerInputEvent gets triggered. Inside the callback, based on what type of PlayerInput it is and what type of Action the player is currently doing, it decides what action the player should now start doing, and what action the player should return to once it finishes the action. I make use of lots of when statements, which are like Java's switch statements. Note that when statements can also be used as expressions, so they can be used inside a function call.

## Actions

In my game, actions are any action related to movement or combat that an entity can do. Each entity can only do one action at a time. Both the player and the enemies can do actions. In this way, controlling the player is reduced to mapping the input to the appropriate action, and the enemy AI is reduced to mapping the enemy's internal state to an appropriate action (I explain enemy AI in detail below).

Actions are represented by the enum Action, defined like this:

enum class Action {

IDLE, WALK, RUN, PUNCH, SPIN\_PUNCH, PUSH, HIT\_KB, UP\_STAIRS, DOWN\_STAIRS

}

The IDLE and RUN actions are used by both the player and the enemies. At the moment only the enemies WALK. Only the player performs PUNCH, SPIN\_PUNCH, UP\_STAIRS and DOWN\_-STAIRS, and only the enemies perform PUSH and HIT\_KB.

The action that an entity is currently doing is stored in its ActionComp, along with various other information:

* runSpeed: the number of world units per second that the entity moves by when it's running
* walkSpeed: the number of world units per second that the entity moves by when it's walking
* action: the action that the entity is currently doing
* facing: the direction that the entity is currently facing in (more on that below)
* actionStartTime: the time it started doing the action
* returnToAction: the action that the entity should return to once it finishes its current action
* actionState: the state of the action (more on that later)

The facing direction is represented by an enum, as you can see below. The syntax is a little difficult to understand, so I will explain it. There are 2 entries in Facing: LEFT and RIGHT. Each entry is required to have an integer that represents the sign of the direction (this is just to simplify code a bit in ActionSys). For LEFT the sign is -1 and for RIGHT the sign is 1. Each entry also has a reverse method, which returns RIGHT when called on LEFT and returns LEFT when called on RIGHT.

enum class Facing(val sign: Int) {

LEFT(-1), RIGHT(1);

fun reverse() = when (this) {

LEFT -> RIGHT

RIGHT -> LEFT

}

}

ActionSys takes care of actually performing these actions. It iterates over every entity with an ActionComp, TransformComp and HitboxComp and it performs the action stored in its Action-Comp. If the action is IDLE, it does nothing. If it's WALK or RUN, it increases the x position by the facing's sign (as described above), times the walk/run speed, times deltaTime. deltaTime is the time it took to completely process the previous frame. Speed = distance / time, so multiplying the speed by deltaTime results in the distance that the entity should move in a single frame. Further-more, I also move the x position of the entity's hitbox by the same amount. The Transform-Comp's position is the position at which the entity is rendered, whereas the Hitbox-Comp's position defines the area in which attacks have to land for damage to be dealt.

when (actionComp.action) {

Action.IDLE -> {}

Action.WALK -> {

transformComp.x += actionComp.facing.sign \* actionComp.walkSpeed \* deltaTime

hitboxComp.x += actionComp.facing.sign \* actionComp.walkSpeed \* deltaTime

}

Action.RUN -> {

transformComp.x += actionComp.facing.sign \* actionComp.runSpeed \* deltaTime

hitboxComp.x += actionComp.facing.sign \* actionComp.runSpeed \* deltaTime

}

Next, I will explain the findClosestNearbyEntityInDirection method (below), which is used for several of the actions. This method is used for finding the closest nearby entity to a given entity in a given direction. It's used for instance to find which entity a punch should damage. The method iterates over all entities in the engine. As explained in the code, it performs various checks on each entity, ignoring entities that don't pass the checks. If the entity passes all the tests, and it is closer to the given entity than the current closest entity (or if the current closest entity is null), the entity is stored in the target variable. Therefore, when it returns target, it returns the closest nearby entity in the given direction, or null if there are no nearby entities.

private fun findClosestNearbyEntityInDirection(entity: Entity, facing: Facing): Entity? {

val hitboxComp = entity.getNotNull(HitboxComp.mapper)

var target: Entity? = null

for (possibleTarget in entities) {

//ignore itself

if (possibleTarget == entity) continue

val otherHitboxComp = possibleTarget[HitboxComp.mapper]

val otherHPComp = possibleTarget[HPComp.mapper]

//ignore entities without a hitbox or without health

if (otherHitboxComp == null || otherHPComp == null) continue

//ignore entities that are too high or too low

if (otherHitboxComp.top < hitboxComp.bottom || otherHitboxComp.bottom > hitboxComp.top) continue

if (facing == Facing.LEFT) {

// ignore entities that are on the right

if (otherHitboxComp.left > hitboxComp.left) continue

//ignore enemies that are too far away to the left

if (otherHitboxComp.right < hitboxComp.left - 20) continue

//store closest valid target

if (target == null) {

target = possibleTarget

}

else {

val targetHitboxComp = target.getNotNull(HitboxComp.mapper)

if (otherHitboxComp.right > targetHitboxComp.right) {

target = possibleTarget

}

}

}

else { //facing right

// ignore entities that are on the left

if (otherHitboxComp.right < hitboxComp.right) continue

//ignore enemies that are too far away to the right

if (otherHitboxComp.left > hitboxComp.right + 20) continue

//store closest valid target

if (target == null) {

target = possibleTarget

}

else {

val targetHitboxComp = target.getNotNull(HitboxComp.mapper)

if (otherHitboxComp.left < targetHitboxComp.left) {

target = possibleTarget

}

}

}

}

return target

}

Continuing with the different actions, if the action is PUNCH, first it checks whether it has been punching for longer than 0.693 seconds, in which case it starts the action that PlayerInputSys told it to go back to after the punch ends. In most cases this is IDLE. Otherwise, it checks if it has been punching for over 0.231 seconds, because that is when the AnimationSys switches to the frame with the arm extended, so that is when the punch's damage should be dealt. (The animation is explained in the next section.) If true, it finds uses the function explained above to find the target entity, decreases its HP by 25, and plays the punch sound effect. Finally, it switches the action's state from 0 to 1 so that this effect only occurs once per punch. SPIN\_PUNCH works similarly.

Action.PUNCH -> {

//stop punching after punch frame ends

if (timeSys.appUptime > actionComp.actionStartTime + 0.693f) {

actionComp.startAction(actionComp.returnToAction, timeSys.appUptime)

}

// perform punch effect when switches to punch frame

else if (actionComp.actionState == 0 && timeSys.appUptime > actionComp.actionStartTime + 0.231f) {

val target = findClosestNearbyEntityInDirection(entity, actionComp.facing)

if (target != null) {

val targetHPComp = target.getNotNull(HPComp.mapper)

targetHPComp.hp -= 25f

sounds.playPunchSoft()

}

actionComp.actionState = 1

}

}

The PUSH action is what an enemy can do to prevent the player from getting past them. All it does is checks if the player's hitbox is overlapping with the enemy's hitbox. If it is, it moves the player away so that it's not. The reason that there is a +/- 1 when moving the player is so that it leaves the hitboxes overlapping by just 1 pixel. Otherwise, the enemy would realise that it's no longer overlapping with the player and run towards the player. This would cause a glitch where the enemy is rapidly switching between the running animation and the pushing animation.

Action.PUSH -> {

val playerTransformComp = player.getNotNull(TransformComp.mapper)

val playerHitboxComp = player.getNotNull(HitboxComp.mapper)

when (actionComp.facing) {

Facing.LEFT -> {

val deltaX = (hitboxComp.x - playerHitboxComp.width) - playerHitboxComp.x

if (deltaX < 0) {

playerHitboxComp.x += deltaX + 1

playerTransformComp.x += deltaX + 1

}

}

Facing.RIGHT -> {

val deltaX = (hitboxComp.x + hitboxComp.width) - playerHitboxComp.x

if (deltaX > 0) {

playerHitboxComp.x += deltaX - 1

playerTransformComp.x += deltaX - 1

}

}

}

}

Finally, the UP\_STAIRS and DOWN\_STAIRS actions simply move the player from the lower level to the upper level, or from the upper level to the lower level.

Action.UP\_STAIRS -> {

transformComp.y = 95f

hitboxComp.y = 98f

actionComp.startAction(Action.IDLE, timeSys.appUptime)

}

Action.DOWN\_STAIRS -> {

transformComp.y = 5f

hitboxComp.y = 8f

actionComp.startAction(Action.IDLE, timeSys.appUptime)

}

## Animation

Each entity that has an animated texture has an AnimationComp. Some entities have multiple animation loops that they switch between depending on what action they're currently doing. These are represented by the class AnimationLoop. Each AnimationLoop consists of a list of frames (textures) that the entity will cycle through indefinitely, together with the duration that each frame in seconds. Inside AnimationComp, I store the list of AnimationLoops that the entity needs. I also store a function that determines the index of the loop that the entity should be on, based on the state of the entity's AIComp, and the action the entity is currently doing. I also store a variable that describes whether the texture should be flipped when the entity is facing to the left. Finally, I store some self-explanatory variables that are used by AnimationSys.

Inside AnimationSys, first I execute determineAnimationLoop to find out the index of the Animation-Loop the entity should be on. Then I check whether the AnimationLoop has changed since the previous frame. If it hasn't, and it's been on that frame for longer than the frame duration, it switches the entity's texture to the next one in the AnimationLoop. If the AnimationLoop has changed, it switches the texture to the first frame in the new loop. Finally, if the AnimationComp indicates that the texture should be flipped when facing left, it sets the flippedHorizontally flag in the GraphicsComp accordingly.

## Enemy AI

Every entity with an AI has an AIComp. Inside it 3 functions are stored. They determine the state, action and facing direction of the entity, based on certain factors. The current state is also stored in AIComp.

AISys is very simple, it simply executes these 3 functions for each entity with an AIComp and an ActionComp, starting the action that is returned.

The actual implementation of the AI for both types of enemy is found in SpawningSys, inside spawnOfficeWorker and spawnOfficeWorkerKB (the "KB" stands for "with keyboard"). Under the configuration of the AIComp, you can see each entity's implementation of the 3 AI-related functions. The 2 types of enemy have very similar implementations. They are pretty self-explanatory: WALK when the player is on a different level, RUN when the player is on the same level, and PUSH/HIT\_KB when overlapping with the player. Always face towards the player, unless the player is on a different level, in which case carry on facing in whatever direction it was previously facing.

# Technical Solution

## Table of Contents

### Android Module

| **Class/File** | **Description** | **Page no.** |
| --- | --- | --- |
| AndroidLauncher | Launches the game on Android. | 50 |
| TouchControlsGestureListener | Handles detecting the touch gesture controls on Android. | 51 |

### Lwjgl3 (Desktop) Module

| **Class/File** | **Description** | **Page no.** |
| --- | --- | --- |
| KeyboardControlsInputProcessor | Handles keyboard and mouse events for player controls on PC. | 54 |
| Lwjgl3Launcher | Launches the game on desktop (LWJGL3). | 57 |

### Core Module

| **Class/File** | **Description** | **Page no.** |
| --- | --- | --- |
| asset.ComboKingFonts | Stores and manages the game's fonts. | 58 |
| asset.ComboKingMusic | Stores and manages the game's music assets. | 59 |
| asset.ComboKingSounds | Stores and manages the game's sound assets. | 60 |
| asset.ComboKingTextures | Stores the game's textures. | 61 |
| comp.ActionComp | Entities with an [ActionComp] can move like a human (e.g. running) and perform combat. | 63 |
| comp.AIComp | Entities with an [AIComp] move and fight according to a rule-based AI. | 65 |
| comp.AnimationComp | Entities with an [AnimationComp] have an animated texture. | 66 |
| comp.ButtonComp | Entities with a [ButtonComp] can be clicked on by the user. | 67 |
| comp.GraphicsComp | Entities with a [GraphicsComp] and a [Transform-Comp] are drawn on the screen. | 68 |
| comp.HitboxComp | Entities with a [HitboxComp] have additional size and position to the one specified by [TransformComp]. | 69 |
| comp.HPComp | Entities with an [HPComp] can take damage and have a finite number of health points. | 70 |
| comp.InfoComp | Stores basic information about an entity that is useful for game logic or debugging. | 71 |
| comp.ScoreComp | The score entity has a [ScoreComp] so it can store the current score. | 72 |
| comp.SliderComp | Used by UI elements that slide between different values. | 73 |
| comp.TextComp | Entities with a [TextComp] can specify some text that will be drawn on the screen. | 74 |
| comp.TransformComp | Entities with a [TransformComp] have a size and position in the game world. | 75 |
| event.GameEvent | An event that can be triggered and responded to. Each event type can store its own data. | 77 |
| event.GameEventManager | Enables listening to and triggering [GameEvent]s. | 78 |
| screen.ComboKingScreen | Base class for all [KtxScreen]s in the game. | 79 |
| screen.GameOverScreen | The screen that's showing just after the player has run out of HP. | 80 |
| screen.GameScreen | The screen that's showing when the game is being played. | 83 |
| screen.MenuScreen | The screen that's showing when the user is navigating the menus. | 86 |
| sys.ActionSys | Acts on entities according to their current action as specified by their [ActionComp]. | 91 |
| sys.AISys | Chooses an action for each entity based on the current game state and the rules specified by the entity's [AIComp]. | 95 |
| sys.AnimationSys | Sets entities' textures according to the rules specified by their [AnimationComp]. | 96 |
| sys.BackgroundSys | Controls the background and door entities | 98 |
| sys.CameraSys | Pans the camera to keep the player entity visible on the screen. | 101 |
| sys.DebugRenderSys | Draw entities' hitboxes (for debugging). | 102 |
| sys.GameOverSys | Handles switching to the [GameOverScreen] when the player runs out of HP. | 103 |
| sys.HPRenderSys | Draws the HP of entities above them. | 104 |
| sys.KillingSys | Deletes entities with a HP of 0 or less, if automatic deletion is enabled in their [HPComp]. | 105 |
| sys.MusicSys | Handles playing the game music. | 106 |
| sys.PlayerInputSys | Resolves any keyboard or touch input events relating to controlling the player. | 107 |
| sys.ScoreSys | Increases the score when certain game events are triggered. | 110 |
| sys.SpawningSys | Spawns the enemy entities automatically. | 111 |
| sys.SpriteRenderSys | Renders the entities on the screen. | 116 |
| sys.TextRenderSys | Draws the text of entities with a [TextComp]. | 118 |
| sys.TimeSys | Keeps track of the app uptime and game uptime to enable timing of various things. | 120 |
| util.Util | Various utility functions. | 121 |
| CKLogger | Wrapper for libKTX [Logger] that logs only the tag and the message, without logging the debug level. | 123 |
| CKPrefs | Manages putting values to and getting values from the persistent storage. | 124 |
| ComboKing | The main class. This is created from a platform-specific launcher to start the app. | 125 |
| UIInputProcessor | Handles touch and key events for UI elements. | 127 |

## Groups

### Group A

| **What I've done** | **Evidence** |
| --- | --- |
| Composition-based ECS architec-ture | component and system packages, and entities def-ined in GameScreen and MainMenuScreen |
| Passing around lambda functions | GameEventManager (each callback is a lambda function), AIComp, AnimationComp, ButtonComp, GameScreen |
| Generic functions | GameEventManager, Entity.getNotNull (in Util file) |
| Complex game events model | GameEventManager |
| 3-module structure (meaning vast majority of code is platform-ag-nostic) | See AndroidLauncher, Lwjgl3Launcher and Combo-King for how the modules speak to each other |
| Complex user-defined algorithms | ActionSys.findClosestNearbyEntityInDirection |
| Decomposition of complex behav-iour into interacting classes that each do simple things | The complicated combat logic is decomposed into the (loosely-coupled) classes TouchControlsGestureList-ener, KeyboardControlsInputProcessor, ActionComp, AIComp, AnimationComp, HPComp, ActionSys, AI-Sys, AnimationSys, KillingSys and SpawningSys |

### Group B

| **What I've done** | **Evidence** |
| --- | --- |
| Map of a set | GameEventManager, see callbacks property |
| Companion objects (static fields) | Each of the components |
| Encapsulation | TransformComp, HitboxComp |
| Nullable types | GraphicsComp, GameEventManager, GameOver-Screen, GameScreen |
| Enum classes | ActionComp file, GameEvent file |
| Sealed classes | GameEvent |

### Group C

| **What I've done** | **Evidence** |
| --- | --- |
| Simple mathematical calculations | TouchControlsGestureListener.pan, CameraSys.update, Util file, ActionSys.processEntity, ScoreSys.update |
| Private properties | TransformComp, CameraSys |
| Getters and setters | TransformComp |
| Casting | TransformComp.setSizeFromTexture, GameScreen.show |

# Testing

## Table of Tests

| **✅** | **No.** | **Timestamp** | **Objective/Description** | **Comment** |
| --- | --- | --- | --- | --- |
| ✅ | 1 | 00:01.50 | The game creates a window when it is started on PC/Mac/Linux | Works as expected |
| ✅ | 2 | 00:17.66 | F11 makes the game fullscreen | Works as expected |
| ✅ | 3 | 00:21.53 | 1.1. The "play" button should start the game | Works as expected |
| ✅ | 4 | 00:29.46 | 1.2. The "settings" button should display the settings | Works as expected |
| ✅ | 5 | 00:36.23 | 1.3. The "credits" button should display the credits | Works as expected |
| ✅ | 6 | 00:40.80 | 1.4. In the main menu, to the right of Jane, there should be a display screen (i.e. in the intelligence headquarters, Jane is near a large screen. It is imagined that there are many of these screens throughout the building which are used to communi-cate messages to everyone) | Present |
| ✅ | 7 | 00:43.80 | 1.5. The buttons should be displayed on this screen | Present |
| ✅ | 8 | 00:46.80 | 1.6. The highscore should also be displayed on this screen | Present |
| ✅ | 9 | 00:58.70 | 2.1. There should be a setting to change SFX volume | Works as expected |
| ✅ | 10 | 00:58.70 | 2.2. There should be a setting to change music volume | Works as expected |
| ✅ | 11 | 02:02.43 | 2.3. There should be a button to return to the main menu | Works as expected |
| ✅ | 12 | 02:18.83 | 3.1. All the credits should be displayed | Present |
| ✅ | 13 | 02:21.83 | 3.2. The user should be able to scroll (unless all the credits fit on the screen) | Works as expected |
| ✅ | 14 | 02:41.96 | 3.3. There should be a button to return to the main menu | Works as expected |
| ✅ | 15 | 02:57.76 | 4.1. There should be the current score displayed at the top | Present |
| ✅ | 16 | 03:05.96 | 4.2. The current health should also be displayed | Present |
| ❌ | 17 | N/A | 4.3. There should be a pause button displayed | Not implemented |
| ❌ | 18 | N/A | 4.4. Pressing the pause button pauses the game logic and animation | Not implemented |
| ❌ | 19 | N/A | 4.5. Pressing the pause button displays a popup | Not implemented |
| ❌ | 20 | N/A | 4.6. The popup indicates that the game is paused | Not implemented |
| ❌ | 21 | N/A | 4.7. The popup includes a button to resume the game | Not implemented |
| ❌ | 22 | N/A | 4.8. The popup includes a button to return to the main menu | Not implemented |
| ❌ | 23 | N/A | 4.9. The resume button resumes game logic and animation and closes the popup | Not implemented |
| ❌ | 24 | N/A | 4.10. The main menu button discards the user's progress in the game and returns to the main menu, without saving the score | Not implemented |
| ✅ | 25 | 03:21.43 | 5.1. Jane should be able to run left and right | Works as expected |
| ❌ | 26 | N/A | 5.2. Jane should be able to jump | Not implemented |
| ✅ | 27 | 03:33.50 | 5.3. Jane should be able to go up or down between the ground and upper level | Works as expected |
| ✅ | 28 | 03:41.20 | 5.4. Jane should not be able to walk through enemies; they should block her path | Works as expected |
| ✅ | 29 | 03:48.73 | 6.1. Jane should be able to punch enemies | Works as expected |
| ❌ | 30 | 03:48.73 | 6.2. There should be at least 3 different types of punches | Only 2 implemented |
| ❌ | 31 | N/A | 6.3. Jane should be able to kick enemies | Not implemented |
| ❌ | 32 | N/A | 6.4. There should be at least 3 different types of kicks | 0 implemented |
| ❌ | 33 | 04:25.23 | 7.1. There should be at least 3 different types of enemy | Only 2 implemented |
| ✅ | 34 | 04:25.23 | 7.2. Each type of enemy should behave noticeably differently | Noticeable difference between "push" enemy and "hit" enemy |
| ✅ | 35 | 05:07.83 | 7.3. The enemies should be controlled by a rule-based AI | Evident intelligent behaviour |
| ✅ | 36 | 05:07.83 | 7.4. The enemies should appear to behave intelligently | Evident intelligent behaviour |
| ✅ | 37 | 05:23.36 | 7.5. Enemies should be able to deal damage to Jane, which decreases her health | Works as expected |
| ✅ | 38 | 05:34.43 | 8.1. The game should render a background behind Jane | Present |
| ❌ | 39 | N/A | 8.2. Staircases, ladders, objects that Jane can climb on top of, enemies and items should appear randomly | Not implemented |
| ❌ | 40 | N/A | 8.3. Staircases, ladders, objects that Jane can climb on top of, enemies and items should not appear too close together or too far apart, making the game unfairly easy or hard | Not implemented |
| ✅ | 41 | 06:08.13 | 8.4. The difficulty of the game should increase the further the player gets – this should be done by making tough enemies appear more often and making everything good for the player appear less often | Works as expected |
| ✅ | 42 | 08:43.16 | 8.5. The score increases when Jane progresses to the right | Works as expected |
| ✅ | 43 | 08:50.23 | 8.6. The score also increases when Jane eliminates enemies | Works as expected |
| ✅ | 44 | 09:16.39 | 9.1. A music loop should be played in the main menu | Works as expected |
| ✅ | 45 | 09:28.59 | 9.2. A different loop should be played in the game | Works as expected |
| ✅ | 46 | 10:02.03 | 9.3. Sound effects should be played whenever it makes sense (e.g. the player or an enemy was hit) | Works as expected |
| ✅ | 47 | 10:16.09 | 10.1. When the player runs out of health, a game over screen should be displayed | Works as expected |
| ✅ | 48 | 10:24.42 | 10.2. The score should be displayed | Present |
| ✅ | 49 | 10:27.76 | 10.3. The highscore should be displayed | Present |
| ✅ | 50 | 10:41.16 | 10.4. There should be a button to play again | Present |
| ✅ | 51 | 10:46.16 | 10.5. There should be a button to return to the main menu | Present |
| ✅ | 52 | 10:51.29 | 10.6. If the user has beaten their highscore, this should be indicated | Works as expected |
| ✅ | 53 | 11:49.79 | 10.7. The highscore should be saved so that it persists when the app is closed | Works as expected |
| ✅ | 54 | 12:12.26 | The game also works on Android | Works as expected |

## Testing Video

The video of the tests can be found at <https://youtu.be/c_sRNlJWJVs>. Below is a sample of frames from the video, showcasing some of the tests.

| **Test no.** | **Image(s)** |
| --- | --- |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 25 |  |
| 29 |  |
| 41 |  |
| 52 |  |

# Evaluation

## Adherence to Requirements

As you can see from the Testing section, my project met the majority of my A-level standard objectives. All the objectives which are ticked were met to a full extent. I did not complete any of the extension objectives since I wanted to focus on the A-level standard ones. Overall, I believe I met all the requirements of what would be expected of my project. Although I did not complete all of my objectives, I think this is because I was too ambitious when I wrote the objectives, not because my project is inadequate. Individually, I don't think the objectives are above A-level standard, but there are so many of them, and it becomes very complicated to put it all together such that the codebase remains manageable and well-written in general. The wide scope of the project added lots of complexity. On top of this there is the fact that having so much functionality makes it very time-consuming to code.

## Challenges

It was not easy to code the game. According to IntelliJ IDEA's Statistic plugin, my project has 3,130 (2,422 non-blank and non-comment) lines of Kotlin code, and Kotlin is a very concise programming language. Although I used libraries such as libGDX that handle a lot of low-level stuff for me, I still have to learn how to use the libraries, which is challenging in and of itself. But the most difficult part overall was figuring out how the classes should be designed so that I can get the behaviour I would like, while keeping the code easy to change, easy to extend, sensible, and easy to understand. In other words, flexible, loosely-coupled and coherent.

## Possible Improvements

Perhaps I could have extended objective 7.4 and made the enemy AI more intelligent, for example, the enemies could work together to surround and trap the player, or they could run away when they're nearly out of HP. The flexible design of my code means that it would not be very difficult to implement this.

In the end I did not have time to implement the ability to pause the game. If I wanted to implement it, I would add a new entity inside GameScreen.show, with a ButtonComp that calls a new function GameScreen.pause. This function would add some new entities to the engine to display the pause overlay and the resume and main menu buttons. To actually pause the game I think it would be as simple as disabling TimeSys, as that would stop all the animation and actions from progressing. However, it should also disable some other systems so that they are not wasting resources doing nothing. The resume button would call a GameScreen.resume function that does the opposite of GameScreen.pause. The main menu button would call game.setScreen().

I didn't implement jumping as I didn't think the game needed it. To add jumping, I could add a PhysicsComp which stores vertical velocity and whether the entity is on the ground or in the air. Then a PhysicsSys could iterate over entities with a PhysicsComp, apply the vertical velocity to TransformComp.y, and apply some constant downwards acceleration to the vertical velocity. If the entity is below the ground it would move it on top of the group and turn on the onGround flag. However, if I wanted the player to be able to jump on top of objects and climb between levels as I described in my analysis, the physics would be a lot more complicated and I would probably be better off using Box2D.

Adding more combat moves such as punches and kicks would be implemented by adding new entries to the Action enum and adding new cases to the when statement in ActionSys. Adding new enemies would be implemented by adding new functions to spawn the enemies to SpawningSys. When the function spawns the enemy it would of course use a different AI and different actions to the other functions that spawn enemies.

Random generation of items and objects could also be implemented inside SpawningSys. It would maintain a list of all the spawnable objects and check against it whenever it spawns a new object to make sure the new object isn't too close to an existing object. If the number of objects gets so large that is causes performance issues, I would remove off-screen entities from the engine but keep them in the list and add them back to the engine if they go back on-screen. This means off-screen entities would not be processed by the engine, resulting in a huge performance increase, but also the side effect that off-screen entities do not move.

## Independent Feedback

TODO

# Code

## android: top-level

### AndroidLauncher

**package** io.github.rolodophone.comboking  
  
**import** android.os.Bundle  
**import** com.badlogic.gdx.backends.android.AndroidApplication  
**import** com.badlogic.gdx.backends.android.AndroidApplicationConfiguration  
  
*/\*\*  
 \* Launches the game on Android.  
 \*/***class** AndroidLauncher : AndroidApplication() {  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
  
 initialize(  
 ComboKing **{** gameEventManager, screenWidth, screenHeight **->** TouchControlsGestureListener(gameEventManager, screenWidth, screenHeight).createGestureDetector() **}**,  
 AndroidApplicationConfiguration().apply **{** useImmersiveMode = **true }** )  
 }  
}

### TouchControlsGestureListener

**package** io.github.rolodophone.comboking  
  
**import** com.badlogic.gdx.input.GestureDetector  
**import** com.badlogic.gdx.math.Vector2  
**import** io.github.rolodophone.comboking.event.GameEvent  
**import** io.github.rolodophone.comboking.event.GameEventManager  
**import** io.github.rolodophone.comboking.event.PlayerInput  
**import** kotlin.math.abs  
  
*/\*\*  
 \* The minimum distance (in pixels) that the cursor has to be dragged for it to register as a swipe  
 \*/***private const val** MIN\_SWIPE\_DISTANCE = 40f  
  
*/\*\*  
 \* Handles detecting the touch gesture controls on Android.  
 \*/***class** TouchControlsGestureListener(  
 **private val** gameEventManager: GameEventManager,  
 **private val** screenWidth: Int,  
 **private val** screenHeight: Int  
): GestureDetector.GestureListener {  
  
 **private var** touchHoldXMove = 0f  
 **private var** touchHoldYMove = 0f  
 **private var** touchHoldXCombat = 0f  
  
 **private var** usingLeftControls = **false  
 private var** usingRightControls = **false** */\*\*  
 \* Returns a customised gesture detector for this listener  
 \*/* **fun** createGestureDetector() = GestureDetector(20f, 0.4f, 1.1f, Integer.MAX\_VALUE.toFloat(), **this**)  
  
  
 *// Each method returns true to indicate that the event shouldn't be passed on to other input processors or false  
 // to indicate that it should be passed on.* **override fun** touchDown(x: Float, y: Float, pointer: Int, button: Int): Boolean {  
 **if** (x < screenWidth/2f) {  
 touchHoldXMove = x  
 touchHoldYMove = y  
 usingLeftControls = **true** }  
 **else** {  
 touchHoldXCombat = x  
 usingRightControls = **true** }  
  
 **return false** }  
  
 **override fun** tap(x: Float, y: Float, count: Int, button: Int): Boolean {  
 **return false** }  
  
 **override fun** longPress(x: Float, y: Float): Boolean {  
 **return false** }  
  
 **override fun** fling(velocityX: Float, velocityY: Float, button: Int): Boolean {  
 **return false** }  
  
 **override fun** pan(x: Float, y: Float, deltaX: Float, deltaY: Float): Boolean {  
 *//LHS controls* **if** (usingLeftControls) {  
 **val** distX = abs(x - touchHoldXMove)  
 **val** dispY = touchHoldYMove - y  
 **val** distY = abs(dispY)  
  
 *//using doors* **if** (distY > MIN\_SWIPE\_DISTANCE && distY > distX) {  
 **if** (dispY < 0) { *//swipe down* touchHoldXMove = x  
 touchHoldYMove = y  
 GameEvent.PlayerInputEvent.input = PlayerInput.DOWN\_STAIRS  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **else** { *//swipe up* touchHoldXMove = x  
 touchHoldYMove = y  
 GameEvent.PlayerInputEvent.input = PlayerInput.UP\_STAIRS  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 }  
 **else if** (x < touchHoldXMove - MIN\_SWIPE\_DISTANCE) {  
 touchHoldXMove = x  
 touchHoldYMove = y  
 GameEvent.PlayerInputEvent.input = PlayerInput.LEFT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **else if** (x > touchHoldXMove + MIN\_SWIPE\_DISTANCE) {  
 touchHoldXMove = x  
 touchHoldYMove = y  
 GameEvent.PlayerInputEvent.input = PlayerInput.RIGHT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 }  
 *// RHS controls* **else** {  
 **if** (x < touchHoldXCombat - MIN\_SWIPE\_DISTANCE) {  
 touchHoldXCombat = x  
 GameEvent.PlayerInputEvent.input = PlayerInput.PUNCH\_LEFT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **else if** (x > touchHoldXCombat + MIN\_SWIPE\_DISTANCE) {  
 touchHoldXCombat = x  
 GameEvent.PlayerInputEvent.input = PlayerInput.PUNCH\_RIGHT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 }  
  
 **return true** }  
  
 **override fun** panStop(x: Float, y: Float, pointer: Int, button: Int): Boolean {  
 **if** (usingLeftControls) {  
 GameEvent.PlayerInputEvent.input = PlayerInput.STOP  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
  
 usingLeftControls = **false** usingRightControls = **false  
  
 return false** }  
  
 **override fun** zoom(initialDistance: Float, distance: Float): Boolean {  
 **return false** }  
  
 **override fun** pinch(  
 initialPointer1: Vector2?,  
 initialPointer2: Vector2?,  
 pointer1: Vector2?,  
 pointer2: Vector2?  
 ): Boolean {  
 **return false** }  
  
 **override fun** pinchStop() {  
 }  
}

## lwjgl3: top-level

### KeyboardControlsInputProcessor

**package** io.github.rolodophone.comboking.lwjgl3  
  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.Input  
**import** com.badlogic.gdx.InputProcessor  
**import** io.github.rolodophone.comboking.event.GameEvent  
**import** io.github.rolodophone.comboking.event.GameEventManager  
**import** io.github.rolodophone.comboking.event.PlayerInput  
  
*/\*\*  
 \* Handles keyboard and mouse events for player controls on PC.  
 \*/***class** KeyboardControlsInputProcessor(  
 **private val** gameEventManager: GameEventManager  
): InputProcessor {  
 *// Each method returns true to indicate that the event shouldn't be passed on to other input processors or false   
 // to indicate that it should be passed on.* **override fun** keyDown(keycode: Int): Boolean {  
 **when** (keycode) {  
 Input.Keys.A -> {  
 GameEvent.PlayerInputEvent.input = PlayerInput.LEFT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 **return true** }  
 Input.Keys.D -> {  
 GameEvent.PlayerInputEvent.input = PlayerInput.RIGHT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 **return true** }  
 Input.Keys.J -> {  
 GameEvent.PlayerInputEvent.input = PlayerInput.PUNCH\_LEFT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 **return true** }  
 Input.Keys.L -> {  
 GameEvent.PlayerInputEvent.input = PlayerInput.PUNCH\_RIGHT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 **return true** }  
 Input.Keys.W -> {  
 GameEvent.PlayerInputEvent.input = PlayerInput.UP\_STAIRS  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 **return true** }  
 Input.Keys.S -> {  
 GameEvent.PlayerInputEvent.input = PlayerInput.DOWN\_STAIRS  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 **return true** }  
 }  
  
 **return false** }  
  
 **override fun** keyUp(keycode: Int): Boolean {  
 **when** (keycode) {  
 Input.Keys.A -> {  
 **if** (Gdx.input.isKeyPressed(Input.Keys.D)) {  
 GameEvent.PlayerInputEvent.input = PlayerInput.RIGHT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **else** {  
 GameEvent.PlayerInputEvent.input = PlayerInput.STOP  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **return true** }  
 Input.Keys.D -> {  
 **if** (Gdx.input.isKeyPressed(Input.Keys.A)) {  
 GameEvent.PlayerInputEvent.input = PlayerInput.LEFT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **else** {  
 GameEvent.PlayerInputEvent.input = PlayerInput.STOP  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **return true** }  
 Input.Keys.W, Input.Keys.S -> {  
 **if** (Gdx.input.isKeyPressed(Input.Keys.A)) {  
 GameEvent.PlayerInputEvent.input = PlayerInput.LEFT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **else if** (Gdx.input.isKeyPressed(Input.Keys.D)){  
 GameEvent.PlayerInputEvent.input = PlayerInput.RIGHT  
 gameEventManager.trigger(GameEvent.PlayerInputEvent)  
 }  
 **return true** }  
 }  
  
 **return false** }  
  
 **override fun** keyTyped(character: Char): Boolean {  
 **return false** }  
  
 **override fun** touchDown(screenX: Int, screenY: Int, pointer: Int, button: Int): Boolean {  
 **return false** }  
  
 **override fun** touchUp(screenX: Int, screenY: Int, pointer: Int, button: Int): Boolean {  
 **return false** }  
  
 **override fun** touchDragged(screenX: Int, screenY: Int, pointer: Int): Boolean {  
 **return false** }  
  
 **override fun** mouseMoved(screenX: Int, screenY: Int): Boolean {  
 **return false** }  
  
 **override fun** scrolled(amountX: Float, amountY: Float): Boolean {  
 **return false** }  
}

### Lwjgl3Launcher

**package** io.github.rolodophone.comboking.lwjgl3  
  
**import** com.badlogic.gdx.backends.lwjgl3.Lwjgl3Application  
**import** com.badlogic.gdx.backends.lwjgl3.Lwjgl3ApplicationConfiguration  
**import** io.github.rolodophone.comboking.ComboKing  
**import** io.github.rolodophone.comboking.WORLD\_HEIGHT  
**import** io.github.rolodophone.comboking.WORLD\_WIDTH  
  
*/\*\*  
 \* Launches the game on desktop (LWJGL3).  
 \*/***fun** main() {  
 Lwjgl3Application(  
 ComboKing **{** gameEventManager, \_, \_ **->** KeyboardControlsInputProcessor(gameEventManager)  
 **}**,  
 Lwjgl3ApplicationConfiguration().apply **{** setTitle(**"Combo King"**)  
  
 *//using 16x9 aspect ratio as although newer phones are often longer, there are many more older phones with  
 //exactly 16x9. Also, most tablets are fatter than 16x9 so 16x9 is like a compromise between phones and  
 //tablets. Plus, most monitors are 16x9 so if you play the game in fullscreen it will look perfect.  
  
 //default 3 pixels per world unit* setWindowedMode(WORLD\_WIDTH \* 3, WORLD\_HEIGHT \* 3)  
  
 setWindowIcon(**"libgdx128.png"**, **"libgdx64.png"**, **"libgdx32.png"**, **"libgdx16.png"**)  
 **}** )  
}

## core: asset

### ComboKingFonts

**package** io.github.rolodophone.comboking.asset  
  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.graphics.g2d.BitmapFont  
**import** com.badlogic.gdx.utils.Disposable  
  
*/\*\*  
 \* Stores and manages the game's fonts.  
 \*/***class** ComboKingFonts: Disposable {  
 **val** visitor = BitmapFont(Gdx.files.internal(**"font/visitor.fnt"**), Gdx.files.internal(**"font/visitor.png"**), **false**)  
  
 **override fun** dispose() {  
 visitor.dispose()  
 }  
}

### ComboKingMusic

**package** io.github.rolodophone.comboking.asset  
  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.audio.Music  
**import** com.badlogic.gdx.utils.Disposable  
  
*/\*\*  
 \* Stores and manages the game's music assets.  
 \*/***class** ComboKingMusic: Disposable {  
 **private val** game: Music = Gdx.audio.newMusic(Gdx.files.internal(**"audio/game.ogg"**))  
 **private val** menu: Music = Gdx.audio.newMusic(Gdx.files.internal(**"audio/menu.ogg"**))  
  
 **init** {  
 game.isLooping = **true** menu.isLooping = **true** }  
  
 **fun** playGame() {  
 menu.stop()  
 game.play()  
 }  
  
 **fun** playMenu() {  
 game.stop()  
 menu.play()  
 }  
  
 **fun** setVolume(volume: Float) {  
 game.volume = volume  
 menu.volume = volume  
 }  
  
 **override fun** dispose() {  
 game.dispose()  
 menu.dispose()  
 }  
}

### ComboKingSounds

**package** io.github.rolodophone.comboking.asset  
  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.audio.Sound  
**import** com.badlogic.gdx.utils.Disposable  
  
*/\*\*  
 \* Stores and manages the game's sound assets.  
 \*/*@Suppress(**"unused"**,**"PropertyName"**)  
**class** ComboKingSounds: Disposable {  
 **private val** hitKb: Sound = Gdx.audio.newSound(Gdx.files.internal(**"audio/hit\_kb.ogg"**))  
 **private val** punchHard: Sound = Gdx.audio.newSound(Gdx.files.internal(**"audio/punch\_hard.ogg"**))  
 **private val** punchSoft: Sound = Gdx.audio.newSound(Gdx.files.internal(**"audio/punch\_soft.ogg"**))  
  
 **private var** volume = 1f  
  
 **fun** playHitKb() = hitKb.play(volume)  
 **fun** playPunchHard() = punchHard.play(volume)  
 **fun** playPunchSoft() = punchSoft.play(volume)  
  
 **fun** setVolume(volume: Float) {  
 **if** (volume **in** 0f..1f) {  
 **this**.volume = volume  
 }  
 }  
  
 **override fun** dispose() {  
 hitKb.dispose()  
 punchHard.dispose()  
 punchSoft.dispose()  
 }  
}

### ComboKingTextures

**package** io.github.rolodophone.comboking.asset  
  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.graphics.g2d.TextureAtlas  
**import** com.badlogic.gdx.utils.Disposable  
  
*/\*\*  
 \* Stores the game's textures.  
 \*/*@Suppress(**"unused"**,**"PropertyName"**)  
**class** ComboKingTextures: Disposable {  
 **private val** graphicsAtlas = TextureAtlas(Gdx.files.internal(**"graphics/sprites.atlas"**))  
  
 **val** btn\_back = loadTextureRegion(**"btn\_back"**)  
 **val** btn\_credits = loadTextureRegion(**"btn\_credits"**)  
 **val** btn\_main\_menu = loadTextureRegion(**"btn\_main\_menu"**)  
 **val** btn\_options = loadTextureRegion(**"btn\_options"**)  
 **val** btn\_play\_again = loadTextureRegion(**"btn\_play\_again"**)  
 **val** btn\_play = loadTextureRegion(**"btn\_play"**)  
 **val** content\_options = loadTextureRegion(**"content\_options"**)  
 **val** door\_closed = loadTextureRegion(**"door\_closed"**)  
 **val** door\_open = loadTextureRegion(**"door\_open"**)  
 **val** game\_bg = loadTextureRegion(**"game\_bg"**)  
 **val** main\_menu\_bg = loadTextureRegion(**"main\_menu\_bg"**)  
 **val** office\_worker\_enter\_door = loadTextureRegion(**"office\_worker\_enter\_door"**)  
 **val** office\_worker\_exit\_door\_kb = loadTextureRegion(**"office\_worker\_exit\_door\_kb"**)  
 **val** office\_worker\_exit\_door = loadTextureRegion(**"office\_worker\_exit\_door"**)  
 **val** office\_worker\_hit\_kb0 = loadTextureRegion(**"office\_worker\_hit\_kb0"**)  
 **val** office\_worker\_hit\_kb1 = loadTextureRegion(**"office\_worker\_hit\_kb1"**)  
 **val** office\_worker\_idle\_kb = loadTextureRegion(**"office\_worker\_idle\_kb"**)  
 **val** office\_worker\_idle = loadTextureRegion(**"office\_worker\_idle"**)  
 **val** office\_worker\_push = loadTextureRegion(**"office\_worker\_push"**)  
 **val** office\_worker\_run0 = loadTextureRegion(**"office\_worker\_run0"**)  
 **val** office\_worker\_run1 = loadTextureRegion(**"office\_worker\_run1"**)  
 **val** office\_worker\_run2 = loadTextureRegion(**"office\_worker\_run2"**)  
 **val** office\_worker\_run3 = loadTextureRegion(**"office\_worker\_run3"**)  
 **val** office\_worker\_run4 = loadTextureRegion(**"office\_worker\_run4"**)  
 **val** office\_worker\_run5 = loadTextureRegion(**"office\_worker\_run5"**)  
 **val** office\_worker\_run6 = loadTextureRegion(**"office\_worker\_run6"**)  
 **val** office\_worker\_run7 = loadTextureRegion(**"office\_worker\_run7"**)  
 **val** office\_worker\_run\_kb0 = loadTextureRegion(**"office\_worker\_run\_kb0"**)  
 **val** office\_worker\_run\_kb1 = loadTextureRegion(**"office\_worker\_run\_kb1"**)  
 **val** office\_worker\_run\_kb2 = loadTextureRegion(**"office\_worker\_run\_kb2"**)  
 **val** office\_worker\_run\_kb3 = loadTextureRegion(**"office\_worker\_run\_kb3"**)  
 **val** office\_worker\_run\_kb4 = loadTextureRegion(**"office\_worker\_run\_kb4"**)  
 **val** office\_worker\_run\_kb5 = loadTextureRegion(**"office\_worker\_run\_kb5"**)  
 **val** office\_worker\_run\_kb6 = loadTextureRegion(**"office\_worker\_run\_kb6"**)  
 **val** office\_worker\_run\_kb7 = loadTextureRegion(**"office\_worker\_run\_kb7"**)  
 **val** player\_enter\_door = loadTextureRegion(**"player\_enter\_door"**)  
 **val** player\_exit\_door = loadTextureRegion(**"player\_exit\_door"**)  
 **val** player\_idle0 = loadTextureRegion(**"player\_idle0"**)  
 **val** player\_idle1 = loadTextureRegion(**"player\_idle1"**)  
 **val** player\_punch0 = loadTextureRegion(**"player\_punch0"**)  
 **val** player\_punch1 = loadTextureRegion(**"player\_punch1"**)  
 **val** player\_run0 = loadTextureRegion(**"player\_run0"**)  
 **val** player\_run1 = loadTextureRegion(**"player\_run1"**)  
 **val** player\_run2 = loadTextureRegion(**"player\_run2"**)  
 **val** player\_run3 = loadTextureRegion(**"player\_run3"**)  
 **val** player\_run4 = loadTextureRegion(**"player\_run4"**)  
 **val** player\_run5 = loadTextureRegion(**"player\_run5"**)  
 **val** player\_run6 = loadTextureRegion(**"player\_run6"**)  
 **val** player\_run7 = loadTextureRegion(**"player\_run7"**)  
 **val** player\_spinning\_punch0 = loadTextureRegion(**"player\_spinning\_punch0"**)  
 **val** player\_spinning\_punch1 = loadTextureRegion(**"player\_spinning\_punch1"**)  
 **val** player\_spinning\_punch2 = loadTextureRegion(**"player\_spinning\_punch2"**)  
 **val** railing = loadTextureRegion(**"railing"**)  
 **val** slider0 = loadTextureRegion(**"slider0"**)  
 **val** slider1 = loadTextureRegion(**"slider1"**)  
 **val** slider2 = loadTextureRegion(**"slider2"**)  
 **val** slider3 = loadTextureRegion(**"slider3"**)  
 **val** slider4 = loadTextureRegion(**"slider4"**)  
  
 **override fun** dispose() {  
 graphicsAtlas.dispose()  
 }  
   
 **private fun** loadTextureRegion(name: String): TextureAtlas.AtlasRegion {  
 **val** region = graphicsAtlas.findRegion(name)  
   
 **if** (region == **null**) {  
 **throw** AssertionError(**"Region $**name **not found in texture atlas."**)  
 }  
   
 **else return** region  
 }  
}

## core: comp

### ActionComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with an* ***[ActionComp]*** *can move like a human (e.g. running) and perform combat.  
 \*/***class** ActionComp: Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<ActionComp>()  
 }  
  
 *//these may be specified when creating the entity. If not specified they'll have the default value defined below* **var** runSpeed = 100f  
 **var** walkSpeed = 50f  
  
 *//these shouldn't be specified when creating the entity, as they will be controlled by ActionSys* **var** action = Action.IDLE  
 **private set  
 var** facing = Facing.RIGHT  
 **var** actionStartTime = 0f  
 **var** returnToAction = Action.IDLE  
 **var** actionState = 0  
  
 **override fun** reset() {  
 runSpeed = 100f  
  
 action = Action.IDLE  
 facing = Facing.RIGHT  
 actionStartTime = 0f  
 returnToAction = Action.IDLE  
 actionState = 0  
 }  
  
 **fun** startAction(value: Action, gameUptime: Float) {  
 **if** (value != action) {  
 actionStartTime = gameUptime  
 action = value  
 actionState = 0  
 }  
 }  
}  
  
**enum class** Action {  
 IDLE, WALK, RUN, PUNCH, SPIN\_PUNCH, PUSH, HIT\_KB, UP\_STAIRS, DOWN\_STAIRS  
}  
  
**enum class** Facing(**val** sign: Int) {  
 LEFT(-1), RIGHT(1);  
  
 **fun** reverse() = **when** (**this**) {  
 LEFT -> RIGHT  
 RIGHT -> LEFT  
 }  
}

### AIComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with an* ***[AIComp]*** *move and fight according to a rule-based AI.  
 \*/***class** AIComp: Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<AIComp>()  
 }  
  
 **var** determineState: (enemy: Entity, player: Entity) -> Int  
 = **{** \_, \_ **->** 0 **}  
 var** determineAction: (enemy: Entity, player: Entity, state: Int) -> Action  
 = **{** \_, \_, \_ **->** Action.IDLE **}  
 var** determineFacing: (enemy: Entity, player: Entity, state: Int) -> Facing  
 = **{** \_, \_, \_ **->** Facing.RIGHT **}  
  
 var** state = 0  
  
 **override fun** reset() {  
 determineState = **{** \_, \_ **->** 0 **}** determineAction = **{** \_, \_, \_ **->** Action.IDLE **}** determineFacing = **{** \_, \_, \_ **->** Facing.RIGHT **}** state = 0  
 }  
}

### AnimationComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with an* ***[AnimationComp]*** *have an animated texture.  
 \*/***class** AnimationComp : Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<AnimationComp>()  
 }  
  
 **class** AnimationLoop(**val** frameDuration: Float, **val** frames: List<TextureRegion>) {  
 **override fun** toString() = **"AnimationLoop(${**frames.joinToString(**", "**)**})"** }  
  
 **var** animationLoops = listOf<AnimationLoop>()  
 **var** determineAnimationLoop: (state: Int?, action: Action?) -> Int = **{** \_, \_ **->** 0 **}  
 var** flipWhenFacingLeft = **true** *//for use of AnimationSys* **var** animationLoop = 0  
 **var** timeOfLastFrameChange = 0f  
 **var** frameIndex = 0  
  
 **override fun** reset() {  
 animationLoops = listOf()  
 determineAnimationLoop = **{** \_, \_ **->** 0 **}** flipWhenFacingLeft = **true** animationLoop = 0  
 timeOfLastFrameChange = 0f  
 frameIndex = 0  
 }  
}

### ButtonComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with a* ***[ButtonComp]*** *can be clicked on by the user.  
 \*/***class** ButtonComp : Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<ButtonComp>()  
 }  
  
 **var** onPress: () -> Unit = **{}  
  
 override fun** reset() {  
 onPress = **{}** }  
}

### GraphicsComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with a* ***[GraphicsComp]*** *and a* ***[TransformComp]*** *are drawn on the screen.  
 \*/***class** GraphicsComp: Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<GraphicsComp>()  
 }  
  
 */\*\*  
 \* The entity's image.  
 \*/* **var** textureRegion: TextureRegion? = **null  
 var** visible = **true  
 var** flippedHorizontally = **false  
  
 override fun** reset() {  
 textureRegion = **null** visible = **true** flippedHorizontally = **false** }  
}

### HitboxComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.math.Rectangle  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with a* ***[HitboxComp]*** *have additional size and position to the one specified by* ***[TransformComp]****.  
 \*  
 \* The* ***[HitboxComp]*** *is used for collision detection whereas the* ***[TransformComp]*** *specifies the size and position of the  
 \* entity's texture.  
 \*/***class** HitboxComp : Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<HitboxComp>()  
 }  
  
 **private val** rect = Rectangle()  
  
 **var** x: Float  
 **set**(value) { rect.x = value }  
 **get**() = rect.x  
  
 **var** y: Float  
 **set**(value) { rect.y = value }  
 **get**() = rect.y  
  
 **var** width: Float  
 **set**(value) { rect.width = value }  
 **get**() = rect.width  
  
 **var** height: Float  
 **set**(value) { rect.height = value }  
 **get**() = rect.height  
  
 **val** left: Float  
 **get**() = x  
  
 **val** bottom: Float  
 **get**() = y  
  
 **val** right: Float  
 **get**() = x + width  
  
 **val** top: Float  
 **get**() = y + height  
  
 **override fun** reset() {  
 rect.set(0f, 0f, 0f, 0f)  
 }  
  
 **fun** overlaps(otherHitboxComp: HitboxComp) = rect.overlaps(otherHitboxComp.rect)  
}

### HPComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with an* ***[HPComp]*** *can take damage and have a finite number of health points.  
 \*/***class** HPComp : Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<HPComp>()  
 }  
  
 **var** deleteWhenHP0 = **false  
 var** hp = 1f  
  
 **override fun** reset() {  
 deleteWhenHP0 = **false** hp = 1f  
 }  
}

### InfoComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Stores basic information about an entity that is useful for game logic or debugging.  
 \*/***class** InfoComp : Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<InfoComp>()  
 }  
  
 **var** tags = mutableListOf<String>()  
 **var** name = **"Unnamed"  
  
 override fun** reset() {  
 tags = mutableListOf()  
 name = **"Unnamed"** }  
}

### ScoreComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* The score entity has a* ***[ScoreComp]*** *so it can store the current score.  
 \*/***class** ScoreComp: Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<ScoreComp>()  
 }  
  
 **var** distance = 0f  
 **var** kills = 0  
 **var** time = 0f  
 **var** score = 0  
  
 **override fun** reset() {  
 distance = 0f  
 kills = 0  
 time = 0f  
 score = 0  
 }  
}

### SliderComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Used by UI elements that slide between different values.  
 \*/***class** SliderComp: Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<SliderComp>()  
 }  
  
 **var** onChange: (progress: Float) -> Unit = **{}  
 var** progress = 0f  
 **var** textures = mutableListOf<TextureRegion>()  
  
 **override fun** reset() {  
 onChange = **{}** progress = 0f  
 }  
}

### TextComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.graphics.Color  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with a* ***[TextComp]*** *can specify some text that will be drawn on the screen.  
 \*  
 \* The coordinates where the text is drawn is specified by a* ***[TransformComp]****. The x and y coordinates define the  
 \* location of the bottom-left corner of the text. The width and height is fixed for any given text.  
 \*/***class** TextComp: Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<TextComp>()  
 **const val** FONT\_HEIGHT = 5f  
 }  
  
 **var** text = **""  
 var** colour: Color = Color.WHITE  
  
 **override fun** reset() {  
 text = **""** colour = Color.WHITE  
 }  
}

### TransformComp

**package** io.github.rolodophone.comboking.comp  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion  
**import** com.badlogic.gdx.math.Rectangle  
**import** com.badlogic.gdx.utils.Pool  
**import** ktx.ashley.mapperFor  
  
*/\*\*  
 \* Entities with a* ***[TransformComp]*** *have a size and position in the game world.  
 \*/***class** TransformComp: Component, Pool.Poolable {  
 **companion object** {  
 **val** mapper = mapperFor<TransformComp>()  
 }  
  
 **private val** rect = Rectangle()  
  
 **var** x: Float  
 **set**(value) { rect.x = value }  
 **get**() = rect.x  
  
 **var** y: Float  
 **set**(value) { rect.y = value }  
 **get**() = rect.y  
  
 **var** width: Float  
 **set**(value) { rect.width = value }  
 **get**() = rect.width  
  
 **var** height: Float  
 **set**(value) { rect.height = value }  
 **get**() = rect.height  
  
 **val** left: Float  
 **get**() = x  
  
 **val** bottom: Float  
 **get**() = y  
  
 **val** right: Float  
 **get**() = x + width  
  
 **val** top: Float  
 **get**() = y + height  
  
 */\*\*  
 \* The sprite will be drawn after all sprites with a lower* ***[z]*** *\*/* **var** z = 0f  
  
 **override fun** reset() {  
 rect.set(0f, 0f, 0f, 0f)  
 z = 0f  
 }  
  
 **fun** setSizeFromTexture(texture: TextureRegion) {  
 rect.setSize(texture.regionWidth.toFloat(), texture.regionHeight.toFloat())  
 }  
  
 **fun** contains(x: Float, y: Float) = rect.contains(x, y)  
  
 **fun** overlaps(otherTransformComp: TransformComp) = rect.overlaps(otherTransformComp.rect)  
 **fun** overlaps(otherHitboxComp: HitboxComp)  
 = **this**.right > otherHitboxComp.left  
 && **this**.left < otherHitboxComp.right  
 && **this**.top > otherHitboxComp.bottom  
 && **this**.bottom < otherHitboxComp.top  
}

## core: event

### GameEvent

**package** io.github.rolodophone.comboking.event  
  
**import** io.github.rolodophone.comboking.comp.ScoreComp  
  
*/\*\*  
 \* An event that can be triggered and responded to. Each event type can store its own data.  
 \*/***sealed class** GameEvent {  
  
 **abstract override fun** toString(): String  
  
 */\*\*  
 \* Triggered by the input processors to communicate an input to the [PlayerInputSys][io.github.rolodophone.comboking  
 \* .sys.PlayerInputSys].  
 \*/* **object** PlayerInputEvent: GameEvent() {  
 **var** input = PlayerInput.STOP  
 **override fun** toString() = **"PlayerInputEvent(${**input.name**})"** }  
  
 **object** GameOverEvent: GameEvent() {  
 **lateinit var** scoreComp: ScoreComp  
 **override fun** toString() = **"GameOverEvent"** }  
}  
  
**enum class** PlayerInput {  
 STOP, LEFT, RIGHT, PUNCH\_LEFT, PUNCH\_RIGHT, UP\_STAIRS, DOWN\_STAIRS  
}

### GameEventManager

**package** io.github.rolodophone.comboking.event  
  
**import** com.badlogic.gdx.utils.ObjectMap  
**import** io.github.rolodophone.comboking.ckLogger  
  
**private val** log = ckLogger<GameEventManager>()  
  
*/\*\*  
 \* Enables listening to and triggering* ***[GameEvent]****s.  
 \*/***class** GameEventManager {  
 **private val** callbacks = ObjectMap<GameEvent, MutableSet<(GameEvent) -> Unit>>()  
  
 */\*\*  
 \* Register the given callback for the given type of* ***[GameEvent]****.  
 \*/* **fun** <E: GameEvent> listen(event: E, callback: (event: E) -> Unit) {  
 **val** callbackSet = callbacks[event]  
  
 *//casting so all values are the same type* @Suppress(**"UNCHECKED\_CAST"**)  
 **val** castedCallback = callback **as** (GameEvent) -> Unit  
  
 **if** (callbackSet == **null**) {  
 *// no callbacks present that take that GameEvent so add a new set* callbacks.put(event, mutableSetOf(castedCallback))  
 }  
 **else** {  
 *// add to the existing set of callbacks that take that GameEvent* callbackSet.add(castedCallback)  
 }  
 }  
  
 */\*\*  
 \* Deregister the given callback for the given type of* ***[GameEvent]****.  
 \*/* **fun** <E: GameEvent> stopListening(event: E, callback: (event: E) -> Unit) {  
 **val** callbackSet = callbacks[event]  
 callbackSet?.remove(callback)  
 }  
  
 */\*\*  
 \* Trigger the registered callbacks for the given* ***[GameEvent]****.  
 \*/* **fun** trigger(event: GameEvent) {  
 log.debug **{ "Event triggered: $**event**" }  
  
 val** listenerSet = callbacks[event]  
  
 **if** (listenerSet != **null**) {  
 **for** (callback **in** callbacks[event]) {  
 callback.invoke(event)  
 }  
 }  
 }  
}

## core: screen

### ComboKingScreen

**package** io.github.rolodophone.comboking.screen  
  
**import** io.github.rolodophone.comboking.ComboKing  
**import** ktx.app.KtxScreen  
  
**private const val** MAX\_DELTA\_TIME = 1/10f  
  
*/\*\*  
 \* Base class for all* ***[KtxScreen]****s in the game.  
 \*/***abstract class** ComboKingScreen(**val** game: ComboKing): KtxScreen {  
 **val** batch = game.batch  
 **val** viewport = game.viewport  
 **val** engine = game.engine  
 **val** textures = game.comboKingTextures  
 **val** fonts = game.comboKingFonts  
 **val** music = game.comboKingMusic  
 **val** sounds = game.comboKingSounds  
 **val** gameEventManager = game.gameEventManager  
  
 **override fun** render(delta: Float) {  
 **val** newDeltaTime = **if** (delta > MAX\_DELTA\_TIME) MAX\_DELTA\_TIME **else** delta  
 engine.update(newDeltaTime)  
 }  
  
 **override fun** resize(width: Int, height: Int) {  
 viewport.update(width, height, **false**)  
 }  
}

### GameOverScreen

**package** io.github.rolodophone.comboking.screen  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.gdx.graphics.Color  
**import** com.badlogic.gdx.graphics.OrthographicCamera  
**import** io.github.rolodophone.comboking.ComboKing  
**import** io.github.rolodophone.comboking.comp.\*  
**import** io.github.rolodophone.comboking.event.GameEvent  
**import** io.github.rolodophone.comboking.sys.TextRenderSys  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.entity  
**import** ktx.ashley.with  
  
*/\*\*  
 \* The screen that's showing just after the player has run out of HP.  
 \*/***class** GameOverScreen(  
 game: ComboKing  
): ComboKingScreen(game) {  
  
 **private lateinit var** scoreEntity: Entity  
 **private var** scoreComp: ScoreComp? = **null  
  
 private lateinit var** textRenderSys: TextRenderSys  
  
 **private val** gameOverEventCallback = **{** event: GameEvent.GameOverEvent **->** *//score score info* scoreComp = ScoreComp().apply **{** distance = event.scoreComp.distance  
 kills = event.scoreComp.kills  
 time = event.scoreComp.time  
 score = event.scoreComp.score  
 **}  
 }  
  
 override fun** show() {  
 *//set camera* with(viewport.camera **as** OrthographicCamera) **{** zoom = 1f  
 position.set(viewport.worldWidth / 2f, viewport.worldHeight / 2f, 0f)  
 update()  
 **}** *//add entities* scoreEntity = engine.entity **{** with<InfoComp> **{** name = **"GameOverScoreText"  
 }** with<TransformComp> **{** x = 80f  
 y = 160f  
 **}** with<TextComp> **{** colour = Color(95/255f, 205/255f, 228/255f, 1f)  
 *//text is set soon after when I receive the GameOverEvent* **}  
 }** engine.entity **{** with<InfoComp> **{** name = **"PlayAgainButton"  
 }** with<TransformComp> **{** width = textures.btn\_play\_again.regionWidth \* 2f  
 height = textures.btn\_play\_again.regionHeight \* 2f  
 x = (viewport.worldWidth - width) / 2f  
 y = 60f  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_play\_again  
 **}** with<ButtonComp> **{** onPress = **{** game.setScreen<GameScreen>() **}  
 }  
 }** engine.entity **{** with<InfoComp> **{** name = **"MainMenuButton"  
 }** with<TransformComp> **{** width = textures.btn\_main\_menu.regionWidth \* 2f  
 height = textures.btn\_main\_menu.regionHeight \* 2f  
 x = (viewport.worldWidth - width) / 2f  
 y = 15f  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_main\_menu  
 **}** with<ButtonComp> **{** onPress = **{** game.setScreen<MenuScreen>() **}  
 }  
 }** textRenderSys = TextRenderSys(batch, viewport, fonts)  
 engine.addSystem(textRenderSys)  
  
 gameEventManager.listen(GameEvent.GameOverEvent, gameOverEventCallback)  
 }  
  
 **override fun** render(delta: Float) {  
 **super**.render(delta)  
  
 *//set score display text from game event* scoreComp?.let **{** scoreComp **->  
 val** scoreTextComp = scoreEntity.getNotNull(TextComp.mapper)  
 **val** highscore = game.ckPrefs.getHighscore()  
  
 scoreTextComp.text = **"DISTANCE: ${**scoreComp.distance.toInt()**} (X50)\n"** +  
 **"KILLS: ${**scoreComp.kills**} (X1000)\n"** +  
 **"TIME: ${**scoreComp.time.toInt()**} (X10)\n"** +  
 **"TOTAL SCORE: ${**scoreComp.score**}\n"** +  
 **"HIGHSCORE: $**highscore**\n"  
  
 if** (scoreComp.score > highscore) {  
 scoreTextComp.text += **"NEW HIGHSCORE!\n"** game.ckPrefs.putHighscore(scoreComp.score)  
 }  
  
 **this**.scoreComp = **null  
 }** }  
  
 **override fun** hide() {  
 engine.removeAllEntities()  
 engine.removeSystem(textRenderSys)  
 gameEventManager.stopListening(GameEvent.GameOverEvent, gameOverEventCallback)  
 }  
}

### GameScreen

**package** io.github.rolodophone.comboking.screen  
  
**import** com.badlogic.ashley.core.EntitySystem  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.InputMultiplexer  
**import** com.badlogic.gdx.InputProcessor  
**import** com.badlogic.gdx.graphics.Color  
**import** com.badlogic.gdx.graphics.OrthographicCamera  
**import** io.github.rolodophone.comboking.ComboKing  
**import** io.github.rolodophone.comboking.WORLD\_HEIGHT  
**import** io.github.rolodophone.comboking.comp.\*  
**import** io.github.rolodophone.comboking.event.GameEventManager  
**import** io.github.rolodophone.comboking.sys.\*  
**import** ktx.ashley.entity  
**import** ktx.ashley.with  
  
*/\*\*  
 \* The screen that's showing when the game is being played.  
 \*/***class** GameScreen(  
 game: ComboKing,  
 **private val** createPlayerInputProcessor: (GameEventManager: GameEventManager, screenWidth: Int, screenHeight: Int)  
 -> InputProcessor  
): ComboKingScreen(game) {  
  
 **private lateinit var** playerInputProcessor: InputProcessor  
  
 **private var** gameScreenSystems: List<EntitySystem>? = **null  
  
 override fun** show() {  
 *//add player controls input processor* playerInputProcessor = createPlayerInputProcessor(gameEventManager, viewport.screenWidth, viewport.screenHeight)  
 (Gdx.input.inputProcessor **as** InputMultiplexer).addProcessor(playerInputProcessor)  
  
 *//set camera* with(viewport.camera **as** OrthographicCamera) **{** zoom = 1f  
 position.set(viewport.worldWidth / 2f, viewport.worldHeight / 2f, 0f)  
 update()  
 **}** *//start counting gameUptime* game.timeSys.gameUptime = 0f  
  
 *// add entities* **val** player = engine.entity **{** with<InfoComp> **{** name = **"Player"  
 }** with<TransformComp> **{** x = 120f  
 y = 5f  
 setSizeFromTexture(textures.player\_idle0)  
 **}** with<GraphicsComp>()  
 with<AnimationComp> **{** animationLoops = listOf(  
 AnimationComp.AnimationLoop(0.462f, listOf(textures.player\_idle0, textures.player\_idle1)),  
 AnimationComp.AnimationLoop(0.077f, listOf(textures.player\_run0, textures.player\_run1,  
 textures.player\_run2, textures.player\_run3, textures.player\_run4, textures.player\_run5,  
 textures.player\_run6, textures.player\_run7)),  
 AnimationComp.AnimationLoop(0.231f, listOf(textures.player\_punch0, textures.player\_punch1,  
 textures.player\_punch0)),  
 AnimationComp.AnimationLoop(0.231f, listOf(textures.player\_spinning\_punch0,  
 textures.player\_spinning\_punch1, textures.player\_spinning\_punch2,  
 textures.player\_spinning\_punch1, textures.player\_spinning\_punch0)),  
 )  
 determineAnimationLoop = **{** \_, action **->  
 when** (action) {  
 Action.IDLE -> 0  
 Action.RUN -> 1  
 Action.PUNCH -> 2  
 Action.SPIN\_PUNCH -> 3  
 Action.UP\_STAIRS, Action.DOWN\_STAIRS -> 0  
 **else** -> **throw** RuntimeException(**"No animation loop for action $**action**"**)  
 }  
 **}** animationLoop = 1 *//set not equal to 0 so that it triggers an AnimationLoop change* **}** with<ActionComp> **{** runSpeed = 120f  
 **}** with<HitboxComp> **{** x = 136f  
 y = 8f  
 width = 16f  
 height = 52f  
 **}** with<HPComp> **{** deleteWhenHP0 = **false** hp = 100f  
 **}  
 }  
 val** scoreEntity = engine.entity **{** with<InfoComp> **{** name = **"Score"  
 }** with<TransformComp> **{** x = 1f  
 y = WORLD\_HEIGHT - TextComp.FONT\_HEIGHT - 1f  
 **}** with<ScoreComp>()  
 with<TextComp> **{** colour = Color.BLACK  
 *//text is set each frame by ScoreSys* **}  
 }** *//add systems* gameScreenSystems = listOf(  
 PlayerInputSys(player, gameEventManager, game.timeSys),  
 AISys(player, game.timeSys),  
 ActionSys(player, sounds, game.timeSys),  
 SpawningSys(player, textures, game.timeSys),  
 KillingSys(scoreEntity),  
 ScoreSys(player, scoreEntity, game.timeSys),  
 BackgroundSys(textures, player),  
 CameraSys(viewport, player),  
 GameOverSys(game, gameEventManager, player, scoreEntity),  
 AnimationSys(game.timeSys),  
 TextRenderSys(batch, viewport, fonts),  
 *//DebugRenderSys(viewport),* HPRenderSys(batch, viewport, fonts)  
 )  
  
 gameScreenSystems?.forEach **{** system **->** engine.addSystem(system)  
 **}** }  
  
 **override fun** hide() {  
 *//stop counting gameUptime* game.timeSys.gameUptime = -1f  
  
 engine.removeAllEntities()  
  
 (Gdx.input.inputProcessor **as** InputMultiplexer).removeProcessor(playerInputProcessor)  
  
 gameScreenSystems?.forEach **{** system **->** engine.removeSystem(system)  
 **}** gameScreenSystems = **null** }  
}

### MenuScreen

**package** io.github.rolodophone.comboking.screen  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.gdx.graphics.Color  
**import** com.badlogic.gdx.graphics.OrthographicCamera  
**import** io.github.rolodophone.comboking.ComboKing  
**import** io.github.rolodophone.comboking.comp.\*  
**import** io.github.rolodophone.comboking.sys.TextRenderSys  
**import** ktx.ashley.entity  
**import** ktx.ashley.with  
  
*/\*\*  
 \* The screen that's showing when the user is navigating the menus.  
 \*/***class** MenuScreen(game: ComboKing): ComboKingScreen(game) {  
 **private val** menuContentEntities = mutableListOf<Entity>()  
  
 **private lateinit var** textRenderSys: TextRenderSys  
  
 **private lateinit var** creditsTextComp: TextComp  
 **private val** credits = listOf(  
 **"Programming\nSergi\nLange-Soler"**,  
 **"Artwork\nSergi\nLange-Soler"**,  
 **"\"Spy Glass\"\nKevin MacLeod\nCC BY 4.0"**,  
 **"\"In a\nHeartbeat\"\nKevin MacLeod\nCC BY 4.0"**,  
 **"\"Plastic\nImpact 1\"\nSophia\_C\nFreesound\nCC BY 3.0"**,  
 **"\"PUNCH-BOXING-\n04.wav\"\nnewagesoup\nFreesound\nCC0 1.0"**,  
 **"\"Punch2.wav\"\nMerrick079\nFreesound\nCC0 1.0"**,  
 **"\"Visitor\"\nBrian Kent"**,  
 **"\"Setback\"\nBrian Kent"** )  
 **private var** creditIndex = 0  
  
 **override fun** show() {  
 *//set camera* with(viewport.camera **as** OrthographicCamera) **{** zoom = 1/2f  
 position.set(viewport.worldWidth \* 1/4f, viewport.worldHeight \* 1/4f, 0f)  
 update()  
 **}** *// background* engine.entity **{** with<InfoComp> **{** name = **"MainMenuBackground"  
 }** with<TransformComp> **{** x = 0f  
 y = 0f  
 width = textures.main\_menu\_bg.regionWidth \* 2f  
 height = textures.main\_menu\_bg.regionHeight \* 2f  
 **}** with<GraphicsComp> **{** textureRegion = textures.main\_menu\_bg  
 **}  
 }** textRenderSys = TextRenderSys(batch, viewport, fonts)  
 engine.addSystem(textRenderSys)  
  
 showMainMenu()  
 }  
  
 **private fun** showMainMenu() {  
 menuContentEntities.forEach **{** engine.removeEntity(it) **}** menuContentEntities.clear()  
  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"HighscoreText"  
 }** with<TransformComp> **{** x = 66f  
 y = 71f  
 **}** with<TextComp> **{** colour = Color(95/255f, 205/255f, 228/255f, 1f)  
 text = **"HS: "** + game.ckPrefs.getHighscore()  
 **}  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"PlayButton"  
 }** with<TransformComp> **{** x = 89f  
 y = 47f  
 setSizeFromTexture(textures.btn\_play)  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_play  
 **}** with<ButtonComp> **{** onPress = **{** game.setScreen<GameScreen>() **}  
 }  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"OptionsButton"  
 }** with<TransformComp> **{** x = 77f  
 y = 29f  
 setSizeFromTexture(textures.btn\_options)  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_options  
 **}** with<ButtonComp> **{** onPress = **{** showOptionsMenu() **}  
 }  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"CreditsButton"  
 }** with<TransformComp> **{** x = 77f  
 y = 11f  
 setSizeFromTexture(textures.btn\_credits)  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_credits  
 **}** with<ButtonComp> **{** onPress = **{** showCreditsMenu() **}  
 }  
 }**)  
 }  
  
 **private fun** showOptionsMenu() {  
 menuContentEntities.forEach **{** engine.removeEntity(it) **}** menuContentEntities.clear()  
  
 **val** sliderTextures = listOf(game.comboKingTextures.slider0, game.comboKingTextures.slider1,  
 game.comboKingTextures.slider2, game.comboKingTextures.slider3, game.comboKingTextures.slider4)  
  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"OptionsContent"  
 }** with<TransformComp> **{** x = 62f  
 y = 8f  
 setSizeFromTexture(textures.content\_options)  
 **}** with<GraphicsComp> **{** textureRegion = textures.content\_options  
 **}  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"BackButton"  
 }** with<TransformComp> **{** x = 65f  
 y = 67f  
 setSizeFromTexture(textures.btn\_back)  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_back  
 **}** with<ButtonComp> **{** onPress = **{** showMainMenu() **}  
 }  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"SFXSlider"  
 }** with<TransformComp> **{** x = 66f  
 y = 41f  
 setSizeFromTexture(textures.slider0)  
 **}** with<GraphicsComp> **{** textureRegion = sliderTextures[(game.ckPrefs.getSFXVolume() \* 4).toInt()]  
 **}** with<SliderComp> **{** textures.addAll(sliderTextures)  
 onChange = **{** progress **->** sounds.setVolume(progress)  
 game.ckPrefs.putSFXVolume(progress)  
 **}  
 }  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"MusicSlider"  
 }** with<TransformComp> **{** x = 66f  
 y = 15f  
 setSizeFromTexture(textures.slider0)  
 **}** with<GraphicsComp> **{** textureRegion = sliderTextures[(game.ckPrefs.getMusicVolume() \* 4).toInt()]  
 **}** with<SliderComp> **{** textures.addAll(sliderTextures)  
 onChange = **{** progress **->** music.setVolume(progress)  
 game.ckPrefs.putMusicVolume(progress)  
 **}  
 }  
 }**)  
 }  
  
 **private fun** showCreditsMenu() {  
 menuContentEntities.forEach **{** engine.removeEntity(it) **}** menuContentEntities.clear()  
  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"BackButton"  
 }** with<TransformComp> **{** x = 65f  
 y = 67f  
 setSizeFromTexture(textures.btn\_back)  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_back  
 **}** with<ButtonComp> **{** onPress = **{** showMainMenu() **}  
 }  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"PrevCreditButton"  
 }** with<TransformComp> **{** x = 118f  
 y = 67f  
 setSizeFromTexture(textures.btn\_back)  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_back  
 **}** with<ButtonComp> **{** onPress = **{** creditIndex -= 1  
 **if** (creditIndex == -1) creditIndex = credits.size - 1  
 creditsTextComp.text = credits[creditIndex]  
 **}  
 }  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"NextCreditButton"  
 }** with<TransformComp> **{** x = 134f  
 y = 67f  
 setSizeFromTexture(textures.btn\_back)  
 **}** with<GraphicsComp> **{** textureRegion = textures.btn\_back  
 flippedHorizontally = **true  
 }** with<ButtonComp> **{** onPress = **{** creditIndex += 1  
 **if** (creditIndex == credits.size) creditIndex = 0  
 creditsTextComp.text = credits[creditIndex]  
 **}  
 }  
 }**)  
 menuContentEntities.add(engine.entity **{** with<InfoComp> **{** name = **"CreditsText"  
 }** with<TransformComp> **{** x = 65f  
 y = 58f  
 **}** creditsTextComp = with **{** colour = Color(95/255f, 205/255f, 228/255f, 1f)  
 text = credits[creditIndex]  
 **}  
 }**)  
 }  
  
 **override fun** hide() {  
 engine.removeAllEntities()  
 engine.removeSystem(textRenderSys)  
 }  
}

## core: sys

### ActionSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.IteratingSystem  
**import** io.github.rolodophone.comboking.asset.ComboKingSounds  
**import** io.github.rolodophone.comboking.comp.\*  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.allOf  
**import** ktx.ashley.get  
  
*/\*\*  
 \* Acts on entities according to their current action as specified by their* ***[ActionComp]****.  
 \*/***class** ActionSys(  
 **private val** player: Entity,  
 **private val** sounds: ComboKingSounds,  
 **private val** timeSys: TimeSys  
) : IteratingSystem(  
 allOf(TransformComp::**class**, ActionComp::**class**, HitboxComp::**class**).get(), 10  
) {  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** transformComp = entity.getNotNull(TransformComp.mapper)  
 **val** hitboxComp = entity.getNotNull(HitboxComp.mapper)  
 **val** actionComp = entity.getNotNull(ActionComp.mapper)  
  
 **when** (actionComp.action) {  
 Action.IDLE -> {}  
 Action.WALK -> {  
 transformComp.x += actionComp.facing.sign \* actionComp.walkSpeed \* deltaTime  
 hitboxComp.x += actionComp.facing.sign \* actionComp.walkSpeed \* deltaTime  
 }  
 Action.RUN -> {  
 transformComp.x += actionComp.facing.sign \* actionComp.runSpeed \* deltaTime  
 hitboxComp.x += actionComp.facing.sign \* actionComp.runSpeed \* deltaTime  
 }  
 Action.PUNCH -> {  
 *//stop punching after punch frame ends* **if** (timeSys.appUptime > actionComp.actionStartTime + 0.693f) {  
 actionComp.startAction(actionComp.returnToAction, timeSys.appUptime)  
 }  
  
 *// perform punch effect when switches to punch frame* **else if** (actionComp.actionState == 0 && timeSys.appUptime > actionComp.actionStartTime + 0.231f) {  
  
 **val** target = findClosestNearbyEntityInDirection(entity, actionComp.facing)  
  
 **if** (target != **null**) {  
 **val** targetHPComp = target.getNotNull(HPComp.mapper)  
 targetHPComp.hp -= 25f  
 sounds.playPunchSoft()  
 }  
  
 actionComp.actionState = 1  
 }  
 }  
 Action.SPIN\_PUNCH -> {  
 *//stop punching after punch frame ends* **if** (timeSys.appUptime > actionComp.actionStartTime + 0.231f\*5) {  
 actionComp.startAction(actionComp.returnToAction, timeSys.appUptime)  
 }  
  
 *// perform punch effect when switches to punch frame* **else if** (actionComp.actionState == 0 && timeSys.appUptime > actionComp.actionStartTime + 0.231f\*2) {  
  
 **val** target = findClosestNearbyEntityInDirection(entity, actionComp.facing.reverse())  
  
 **if** (target != **null**) {  
 **val** targetHPComp = target.getNotNull(HPComp.mapper)  
 targetHPComp.hp -= 50f  
 sounds.playPunchHard()  
 }  
  
 actionComp.actionState = 1  
 }  
 }  
 Action.PUSH -> {  
 **val** playerTransformComp = player.getNotNull(TransformComp.mapper)  
 **val** playerHitboxComp = player.getNotNull(HitboxComp.mapper)  
  
 **when** (actionComp.facing) {  
 Facing.LEFT -> {  
 **val** deltaX = (hitboxComp.x - playerHitboxComp.width) - playerHitboxComp.x  
 **if** (deltaX < 0) {  
 playerHitboxComp.x += deltaX + 1  
 playerTransformComp.x += deltaX + 1  
 }  
 }  
 Facing.RIGHT -> {  
 **val** deltaX = (hitboxComp.x + hitboxComp.width) - playerHitboxComp.x  
 **if** (deltaX > 0) {  
 playerHitboxComp.x += deltaX - 1  
 playerTransformComp.x += deltaX - 1  
 }  
 }  
 }  
 }  
 Action.HIT\_KB -> {  
 *// perform effect when switches to hit frame* **if** (timeSys.appUptime > actionComp.actionStartTime + 0.462f \* (actionComp.actionState\*2 + 1)) {  
  
 **val** target = player  
  
 **val** targetHPComp = target.getNotNull(HPComp.mapper)  
 targetHPComp.hp -= 10f  
  
 sounds.playHitKb()  
  
 actionComp.actionState++  
 }  
 }  
 Action.UP\_STAIRS -> {  
 transformComp.y = 95f  
 hitboxComp.y = 98f  
 actionComp.startAction(Action.IDLE, timeSys.appUptime)  
 }  
 Action.DOWN\_STAIRS -> {  
 transformComp.y = 5f  
 hitboxComp.y = 8f  
 actionComp.startAction(Action.IDLE, timeSys.appUptime)  
 }  
 }  
 }  
  
 **private fun** findClosestNearbyEntityInDirection(entity: Entity, facing: Facing): Entity? {  
 **val** hitboxComp = entity.getNotNull(HitboxComp.mapper)  
  
 **var** target: Entity? = **null  
  
 for** (possibleTarget **in** entities) {  
 *//ignore itself* **if** (possibleTarget == entity) **continue  
  
 val** otherHitboxComp = possibleTarget[HitboxComp.mapper]  
 **val** otherHPComp = possibleTarget[HPComp.mapper]  
  
 *//ignore entities without a hitbox or without health* **if** (otherHitboxComp == **null** || otherHPComp == **null**) **continue** *//ignore entities that are too high or too low* **if** (otherHitboxComp.top < hitboxComp.bottom || otherHitboxComp.bottom > hitboxComp.top) **continue  
  
 if** (facing == Facing.LEFT) {  
 *// ignore entities that are on the right* **if** (otherHitboxComp.left > hitboxComp.left) **continue** *//ignore enemies that are too far away to the left* **if** (otherHitboxComp.right < hitboxComp.left - 20) **continue** *//store closest valid target* **if** (target == **null**) {  
 target = possibleTarget  
 }  
 **else** {  
 **val** targetHitboxComp = target.getNotNull(HitboxComp.mapper)  
 **if** (otherHitboxComp.right > targetHitboxComp.right) {  
 target = possibleTarget  
 }  
 }  
 }  
 **else** { *//facing right  
 // ignore entities that are on the left* **if** (otherHitboxComp.right < hitboxComp.right) **continue** *//ignore enemies that are too far away to the right* **if** (otherHitboxComp.left > hitboxComp.right + 20) **continue** *//store closest valid target* **if** (target == **null**) {  
 target = possibleTarget  
 }  
 **else** {  
 **val** targetHitboxComp = target.getNotNull(HitboxComp.mapper)  
 **if** (otherHitboxComp.left < targetHitboxComp.left) {  
 target = possibleTarget  
 }  
 }  
 }  
 }  
  
 **return** target  
 }  
}

### AISys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.IteratingSystem  
**import** io.github.rolodophone.comboking.comp.AIComp  
**import** io.github.rolodophone.comboking.comp.ActionComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.allOf  
  
*/\*\*  
 \* Chooses an action for each entity based on the current game state and the rules specified by the entity's* ***[AIComp]****.  
 \*/***class** AISys(  
 **private val** player: Entity,  
 **private val** timeSys: TimeSys  
): IteratingSystem(  
 allOf(AIComp::**class**, ActionComp::**class**).get(), 10  
) {  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** entityAIComp = entity.getNotNull(AIComp.mapper)  
 **val** entityMoveComp = entity.getNotNull(ActionComp.mapper)  
  
 entityAIComp.state = entityAIComp.determineState(entity, player)  
 entityMoveComp.startAction(entityAIComp.determineAction(entity, player, entityAIComp.state), timeSys.appUptime)  
 entityMoveComp.facing = entityAIComp.determineFacing(entity, player, entityAIComp.state)  
 }  
}

### AnimationSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.IteratingSystem  
**import** io.github.rolodophone.comboking.ckLogger  
**import** io.github.rolodophone.comboking.comp.\*  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** io.github.rolodophone.comboking.util.repr  
**import** ktx.ashley.allOf  
**import** ktx.ashley.get  
  
**private val** log = ckLogger<AnimationSys>()  
  
*/\*\*  
 \* Sets entities' textures according to the rules specified by their* ***[AnimationComp]****.  
 \*/***class** AnimationSys(  
 **private val** timeSys: TimeSys  
) : IteratingSystem(  
 allOf(AnimationComp::**class**, GraphicsComp::**class**).get(), 20  
) {  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** animationComp = entity.getNotNull(AnimationComp.mapper)  
 **val** graphicsComp = entity.getNotNull(GraphicsComp.mapper)  
 **val** aiComp = entity[AIComp.mapper]  
 **val** actionComp = entity[ActionComp.mapper]  
  
 *//determine the animation loop we should be on* **val** newAnimationLoopIndex = animationComp.determineAnimationLoop(  
 aiComp?.state,  
 actionComp?.action  
 )  
  
 **val** animLoop = animationComp.animationLoops[newAnimationLoopIndex]  
  
 **if** (newAnimationLoopIndex == animationComp.animationLoop) {  
 *//animation loop hasn't changed* **if** (animLoop.frameDuration != -1f &&  
 timeSys.appUptime > animationComp.timeOfLastFrameChange + animLoop.frameDuration  
 ) {  
 *//frame increment is due* **val** numFrameIncrements = ((timeSys.appUptime - animationComp.timeOfLastFrameChange) /  
 animLoop.frameDuration).toInt()  
 animationComp.frameIndex = (animationComp.frameIndex + numFrameIncrements) % animLoop.frames.count()  
 animationComp.timeOfLastFrameChange += numFrameIncrements \* animLoop.frameDuration  
 graphicsComp.textureRegion = animLoop.frames[animationComp.frameIndex]  
 }  
 }  
 **else** {  
 *//animation loop has changed* animationComp.frameIndex = 0  
 animationComp.timeOfLastFrameChange = timeSys.appUptime  
 animationComp.animationLoop = newAnimationLoopIndex  
 graphicsComp.textureRegion = animLoop.frames[0]  
 **if** (entity[InfoComp.mapper]?.name == **"Player"**) {  
 log.debug **{ "${**entity.repr()**} changed animation to$**animLoop**" }** }  
 }  
  
 **if** (animationComp.flipWhenFacingLeft && actionComp != **null**) {  
 graphicsComp.flippedHorizontally = **when** (actionComp.facing) {  
 Facing.LEFT -> **true** Facing.RIGHT -> **false** }  
 }  
 }  
}

### BackgroundSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.core.EntitySystem  
**import** io.github.rolodophone.comboking.asset.ComboKingTextures  
**import** io.github.rolodophone.comboking.comp.GraphicsComp  
**import** io.github.rolodophone.comboking.comp.InfoComp  
**import** io.github.rolodophone.comboking.comp.TransformComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.entity  
**import** ktx.ashley.with  
  
*/\*\*  
 \* Controls the background and door entities  
 \*/***class** BackgroundSys(  
 **private val** textures: ComboKingTextures,  
 **private val** player: Entity  
): EntitySystem(20) {  
  
 **private lateinit var** walls: List<Entity>  
 **private lateinit var** doors: List<Pair<Entity, Entity>>  
 **private lateinit var** railings: List<Entity>  
 **private lateinit var** allEntities: List<List<Entity>>  
 **private var** leftmostIndex = 0  
  
 **override fun** addedToEngine(engine: Engine) {  
 *// create 3 background wall entities, one directly in line with the screen and one on either side* walls = List(3) **{** i **->** engine.entity **{** with<InfoComp> **{** name = **"GameBackground"  
 }** with<TransformComp> **{** x = (textures.game\_bg.regionWidth \* (i-1)).toFloat()  
 y = 0f  
 z = -10f  
 setSizeFromTexture(textures.game\_bg)  
 **}** with<GraphicsComp> **{** textureRegion = textures.game\_bg  
 **}  
 }  
 }** doors = List(3) **{** i **->** Pair(  
 spawnDoor(textures.game\_bg.regionWidth \* (i-1) + 200f, 20f),  
 spawnDoor(textures.game\_bg.regionWidth \* (i-1) + 200f, 110f),  
 )  
 **}** railings = List(3) **{** i **->** engine.entity **{** with<InfoComp> **{** name = **"Railing"  
 }** with<TransformComp> **{** x = (textures.game\_bg.regionWidth \* (i-1)).toFloat()  
 y = 91f  
 z = 10f  
 setSizeFromTexture(textures.railing)  
 **}** with<GraphicsComp> **{** textureRegion = textures.railing  
 **}  
 }  
 }** allEntities = List(3) **{** i **->** listOf(walls[i], doors[i].first, doors[i].second, railings[i])  
 **}** }  
  
 **private fun** spawnDoor(x: Float, y: Float): Entity {  
 **return** engine.entity **{** with<InfoComp> **{** name = **"Door"  
 }** with<TransformComp> **{  
 this**.x = x  
 **this**.y = y  
 **this**.z = -5f  
 setSizeFromTexture(textures.door\_closed)  
 **}** with<GraphicsComp> **{** textureRegion = textures.door\_closed  
 **}  
 }** }  
  
 **override fun** update(deltaTime: Float) {  
 **val** playerTransform = player.getNotNull(TransformComp.mapper)  
  
 **val** middleIndex = (leftmostIndex + 1) % 3  
 **val** rightmostIndex = (leftmostIndex + 2) % 3  
  
 **val** middleWallTransform = walls[middleIndex].getNotNull(TransformComp.mapper)  
 **val** rightmostWallTransform = walls[rightmostIndex].getNotNull(TransformComp.mapper)  
  
 **if** (playerTransform.x < middleWallTransform.x) {  
 *// move rightmost entities to the left-hand side* **for** (entity **in** allEntities[rightmostIndex]) {  
 entity.getNotNull(TransformComp.mapper).x -= textures.game\_bg.regionWidth \* 3  
 }  
 leftmostIndex = rightmostIndex  
 }  
 **else if** (playerTransform.x > rightmostWallTransform.x) {  
 *// move leftmost entities to the right-hand side* **for** (entity **in** allEntities[leftmostIndex]) {  
 entity.getNotNull(TransformComp.mapper).x += textures.game\_bg.regionWidth \* 3  
 }  
 leftmostIndex = middleIndex  
 }  
 }  
}

### CameraSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.core.EntitySystem  
**import** com.badlogic.gdx.utils.viewport.Viewport  
**import** io.github.rolodophone.comboking.comp.TransformComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
  
  
*/\*\*  
 \* Pans the camera to keep the player entity visible on the screen.  
 \*/***class** CameraSys(  
 **private val** viewport: Viewport,  
 **private val** player: Entity  
): EntitySystem(20) {  
  
 **companion object** {  
 **const val** PLAYER\_TRACK\_LEFT\_MARGIN = 120f  
 **const val** PLAYER\_TRACK\_RIGHT\_MARGIN = 120f  
 }  
  
 **override fun** update(deltaTime: Float) {  
 **val** playerTransformComp = player.getNotNull(TransformComp.mapper)  
  
 **if** (viewport.camera.position.x > playerTransformComp.x  
 + viewport.worldWidth / 2f - PLAYER\_TRACK\_LEFT\_MARGIN) {  
 *// player is too close to left side of screen* viewport.camera.position.x = (playerTransformComp.x  
 + viewport.worldWidth / 2f - PLAYER\_TRACK\_LEFT\_MARGIN)  
 }  
 **else if** (viewport.camera.position.x < playerTransformComp.x + playerTransformComp.width  
 - viewport.worldWidth / 2f + PLAYER\_TRACK\_RIGHT\_MARGIN) {  
 *// player is too close to right side of screen* viewport.camera.position.x = (playerTransformComp.x + playerTransformComp.width  
 - viewport.worldWidth / 2f + PLAYER\_TRACK\_RIGHT\_MARGIN)  
 }  
 }  
}

### DebugRenderSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.IteratingSystem  
**import** com.badlogic.gdx.graphics.Color  
**import** com.badlogic.gdx.graphics.glutils.ShapeRenderer  
**import** com.badlogic.gdx.utils.viewport.Viewport  
**import** io.github.rolodophone.comboking.comp.HitboxComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.allOf  
**import** ktx.graphics.use  
  
*/\*\*  
 \* Draw entities' hitboxes (for debugging).  
 \*/***class** DebugRenderSys(  
 **private val** viewport: Viewport  
) : IteratingSystem(  
 allOf(HitboxComp::**class**).get(), 35  
) {  
 **private lateinit var** shapeRenderer: ShapeRenderer  
  
 **override fun** addedToEngine(engine: Engine) {  
 shapeRenderer = ShapeRenderer()  
 **super**.addedToEngine(engine)  
 }  
  
 **override fun** update(deltaTime: Float) {  
 shapeRenderer.color = Color.BLACK  
  
 shapeRenderer.use(ShapeRenderer.ShapeType.Line, viewport.camera.combined) **{  
 super**.update(deltaTime)  
 **}** }  
  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** hitboxComp = entity.getNotNull(HitboxComp.mapper)  
 shapeRenderer.rect(hitboxComp.x, hitboxComp.y, hitboxComp.width, hitboxComp.height)  
 }  
}

### GameOverSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.core.EntitySystem  
**import** io.github.rolodophone.comboking.ComboKing  
**import** io.github.rolodophone.comboking.comp.HPComp  
**import** io.github.rolodophone.comboking.comp.ScoreComp  
**import** io.github.rolodophone.comboking.event.GameEvent  
**import** io.github.rolodophone.comboking.event.GameEventManager  
**import** io.github.rolodophone.comboking.screen.GameOverScreen  
**import** io.github.rolodophone.comboking.util.getNotNull  
  
*/\*\*  
 \* Handles switching to the* ***[GameOverScreen]*** *when the player runs out of HP.  
 \*/***class** GameOverSys(  
 **private val** game: ComboKing,  
 **private val** gameEventManager: GameEventManager,  
 **private val** player: Entity,  
 **private val** scoreEntity: Entity  
) : EntitySystem(20) {  
 **override fun** update(deltaTime: Float) {  
 **val** playerHPComp = player.getNotNull(HPComp.mapper)  
  
 **if** (playerHPComp.hp <= 0) {  
 game.setScreen<GameOverScreen>()  
 GameEvent.GameOverEvent.scoreComp = scoreEntity.getNotNull(ScoreComp.mapper)  
 gameEventManager.trigger(GameEvent.GameOverEvent)  
 }  
 }  
}

### HPRenderSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.IteratingSystem  
**import** com.badlogic.gdx.graphics.Color  
**import** com.badlogic.gdx.graphics.g2d.Batch  
**import** com.badlogic.gdx.utils.viewport.Viewport  
**import** io.github.rolodophone.comboking.asset.ComboKingFonts  
**import** io.github.rolodophone.comboking.comp.HPComp  
**import** io.github.rolodophone.comboking.comp.HitboxComp  
**import** io.github.rolodophone.comboking.comp.TextComp  
**import** io.github.rolodophone.comboking.comp.TransformComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.allOf  
**import** ktx.ashley.get  
**import** ktx.graphics.use  
**import** kotlin.math.roundToInt  
  
*/\*\*  
 \* Draws the HP of entities above them.  
 \*/***class** HPRenderSys(  
 **private val** batch: Batch,  
 **private val** viewport: Viewport,  
 **private val** fonts: ComboKingFonts  
) : IteratingSystem(  
 allOf(TransformComp::**class**, HPComp::**class**).get(), 35  
) {  
 **override fun** update(deltaTime: Float) {  
 fonts.visitor.setUseIntegerPositions(**false**)  
 fonts.visitor.color = Color.BLACK  
  
 batch.use(viewport.camera.combined) **{  
 super**.update(deltaTime)  
 **}** }  
  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** transformComp = entity.getNotNull(TransformComp.mapper)  
 **val** hpComp = entity.getNotNull(HPComp.mapper)  
 **val** hitboxComp = entity[HitboxComp.mapper]  
  
 **if** (hitboxComp == **null**) {  
 fonts.visitor.draw(batch, hpComp.hp.roundToInt().toString(),  
 transformComp.x, transformComp.y + transformComp.height + TextComp.FONT\_HEIGHT + 1f)  
 }  
 **else** {  
 fonts.visitor.draw(batch, hpComp.hp.roundToInt().toString(),  
 hitboxComp.x, hitboxComp.y + hitboxComp.height + TextComp.FONT\_HEIGHT + 1f)  
 }  
 }  
}

### KillingSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.IteratingSystem  
**import** io.github.rolodophone.comboking.comp.HPComp  
**import** io.github.rolodophone.comboking.comp.ScoreComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.allOf  
  
*/\*\*  
 \* Deletes entities with a HP of 0 or less, if automatic deletion is enabled in their* ***[HPComp]****.  
 \*/***class** KillingSys(  
 **private val** scoreEntity: Entity  
): IteratingSystem(  
 allOf(HPComp::**class**).get(), 15  
) {  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** hpComp = entity.getNotNull(HPComp.mapper)  
  
 **if** (hpComp.deleteWhenHP0 && hpComp.hp <= 0) {  
 scoreEntity.getNotNull(ScoreComp.mapper).kills++  
 engine.removeEntity(entity)  
 }  
 }  
}

### MusicSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.EntitySystem  
**import** io.github.rolodophone.comboking.CKPrefs  
**import** io.github.rolodophone.comboking.ComboKing  
**import** io.github.rolodophone.comboking.asset.ComboKingMusic  
**import** io.github.rolodophone.comboking.screen.ComboKingScreen  
**import** io.github.rolodophone.comboking.screen.GameScreen  
  
*/\*\*  
 \* Handles playing the game music.  
 \*/***class** MusicSys(  
 **private val** music: ComboKingMusic,  
 **private val** game: ComboKing,  
 **private val** ckPrefs: CKPrefs  
) : EntitySystem(40) {  
  
 **private var** prevScreen: ComboKingScreen? = **null  
  
 override fun** addedToEngine(engine: Engine) {  
 music.setVolume(ckPrefs.getMusicVolume())  
 }  
  
 **override fun** update(deltaTime: Float) {  
 **if** (game.shownScreen != prevScreen) {  
 **when** (game.shownScreen) {  
 **is** GameScreen -> music.playGame()  
 **else** -> music.playMenu()  
 }  
  
 prevScreen = game.shownScreen  
 }  
 }  
}

### PlayerInputSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.core.EntitySystem  
**import** io.github.rolodophone.comboking.comp.\*  
**import** io.github.rolodophone.comboking.event.GameEvent  
**import** io.github.rolodophone.comboking.event.GameEventManager  
**import** io.github.rolodophone.comboking.event.PlayerInput  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.get  
  
*/\*\*  
 \* Resolves any keyboard or touch input events relating to controlling the player.  
 \*/***class** PlayerInputSys(  
 **private val** player: Entity,  
 **private val** gameEventManager: GameEventManager,  
 **private val** timeSys: TimeSys  
) : EntitySystem(0) {  
  
 **private val** eventCallback = **{** event: GameEvent.PlayerInputEvent **->  
 val** actionComp = player.getNotNull(ActionComp.mapper)  
  
 **when** (event.input) {  
 PlayerInput.STOP -> {  
 **when** (actionComp.action) {  
 Action.RUN -> {  
 actionComp.startAction(Action.IDLE, timeSys.appUptime)  
 }  
 Action.PUNCH, Action.SPIN\_PUNCH, Action.HIT\_KB, Action.PUSH -> {  
 actionComp.returnToAction = Action.IDLE  
 }  
 **else** -> {}  
 }  
 }  
 PlayerInput.LEFT, PlayerInput.RIGHT -> {  
 actionComp.startAction(Action.RUN, timeSys.appUptime)  
 }  
 PlayerInput.PUNCH\_LEFT -> {  
 **when** (actionComp.action) {  
 Action.IDLE -> {  
 actionComp.returnToAction = Action.IDLE  
  
 actionComp.startAction(**when** (actionComp.facing) {  
 Facing.LEFT -> Action.PUNCH  
 Facing.RIGHT -> Action.SPIN\_PUNCH  
 }, timeSys.appUptime)  
 }  
 Action.RUN -> {  
 actionComp.returnToAction = Action.RUN  
  
 actionComp.startAction(**when** (actionComp.facing) {  
 Facing.LEFT -> Action.PUNCH  
 Facing.RIGHT -> Action.SPIN\_PUNCH  
 }, timeSys.appUptime)  
 }  
 **else** -> {}  
 }  
 }  
 PlayerInput.PUNCH\_RIGHT -> {  
 **when** (actionComp.action) {  
 Action.IDLE -> {  
 actionComp.returnToAction = Action.IDLE  
  
 actionComp.startAction(**when** (actionComp.facing) {  
 Facing.LEFT -> Action.SPIN\_PUNCH  
 Facing.RIGHT -> Action.PUNCH  
 }, timeSys.appUptime)  
 }  
 Action.RUN -> {  
 actionComp.returnToAction = Action.RUN  
  
 actionComp.startAction(**when** (actionComp.facing) {  
 Facing.LEFT -> Action.SPIN\_PUNCH  
 Facing.RIGHT -> Action.PUNCH  
 }, timeSys.appUptime)  
 }  
 **else** -> {}  
 }  
 }  
 PlayerInput.UP\_STAIRS -> {  
 **if** (nearDoor(player)) {  
 actionComp.startAction(Action.UP\_STAIRS, timeSys.appUptime)  
 }  
 }  
 PlayerInput.DOWN\_STAIRS -> {  
 **if** (nearDoor(player)) {  
 actionComp.startAction(Action.DOWN\_STAIRS, timeSys.appUptime)  
 }  
 }  
 }  
  
 **when** (event.input) {  
 PlayerInput.LEFT -> actionComp.facing = Facing.LEFT  
 PlayerInput.RIGHT -> actionComp.facing = Facing.RIGHT  
 **else** -> {} *// otherwise keep it the same* }  
 **}  
   
 override fun** addedToEngine(engine: Engine) {  
 gameEventManager.listen(GameEvent.PlayerInputEvent, eventCallback)  
 }  
  
 **override fun** removedFromEngine(engine: Engine) {  
 gameEventManager.stopListening(GameEvent.PlayerInputEvent, eventCallback)  
 }  
  
 **private fun** nearDoor(entity: Entity): Boolean {  
 **val** thisHitboxComp = entity.getNotNull(HitboxComp.mapper)  
  
 **for** (possibleDoor **in** engine.entities) {  
  
 *//ignore entities that are not doors* **if** (possibleDoor[InfoComp.mapper]?.name != **"Door"**) **continue  
  
 val** possibleDoorTransformComp = possibleDoor[TransformComp.mapper] ?: **continue  
  
 if** (possibleDoorTransformComp.overlaps(thisHitboxComp)) **return true** }  
  
 **return false** }  
}

### ScoreSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.core.EntitySystem  
**import** io.github.rolodophone.comboking.comp.ScoreComp  
**import** io.github.rolodophone.comboking.comp.TextComp  
**import** io.github.rolodophone.comboking.comp.TransformComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
  
*/\*\*  
 \* Increases the score when certain game events are triggered.  
 \*/***class** ScoreSys(  
 **private val** player: Entity,  
 **private val** scoreEntity: Entity,  
 **private val** timeSys: TimeSys  
) : EntitySystem(10) {  
 **override fun** update(deltaTime: Float) {  
 **val** scoreComp = scoreEntity.getNotNull(ScoreComp.mapper)  
 **val** scoreTextComp = scoreEntity.getNotNull(TextComp.mapper)  
 **val** playerTransformComp = player.getNotNull(TransformComp.mapper)  
  
 *//calculate score* **val** newDist = (playerTransformComp.x - 120) / 50f  
 **if** (newDist > scoreComp.distance) scoreComp.distance = newDist  
 scoreComp.time = timeSys.gameUptime  
 scoreComp.score = (scoreComp.distance\*50).toInt() +  
 scoreComp.kills\*1000 +  
 (scoreComp.time\*10).toInt()  
  
 scoreTextComp.text = **"SCORE: ${**scoreComp.score**}"** }  
}

### SpawningSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.core.EntitySystem  
**import** io.github.rolodophone.comboking.WORLD\_WIDTH  
**import** io.github.rolodophone.comboking.asset.ComboKingTextures  
**import** io.github.rolodophone.comboking.comp.\*  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** io.github.rolodophone.comboking.util.nextFloat  
**import** ktx.ashley.entity  
**import** ktx.ashley.with  
**import** kotlin.math.abs  
**import** kotlin.random.Random.Default.nextBoolean  
**import** kotlin.random.Random.Default.nextInt  
  
*/\*\*  
 \* Spawns the enemy entities automatically.  
 \*/***class** SpawningSys(  
 **private val** player: Entity,  
 **private val** textures: ComboKingTextures,  
 **private val** timeSys: TimeSys  
): EntitySystem(15) {  
  
 **private lateinit var** playerTransformComp: TransformComp  
  
 **private var** officeWorkerSpawnInterval = 1f  
 **private var** lastOfficeWorkerSpawnTime = 0f  
  
 **override fun** addedToEngine(engine: Engine) {  
 playerTransformComp = player.getNotNull(TransformComp.mapper)  
 }  
  
 **override fun** update(deltaTime: Float) {  
 *//office worker (both with and without keyboard)* **if** (timeSys.appUptime > lastOfficeWorkerSpawnTime + officeWorkerSpawnInterval) {  
  
 **if** (nextBoolean()) spawnOfficeWorker() **else** spawnOfficeWorkerKB()  
  
 lastOfficeWorkerSpawnTime = timeSys.appUptime  
  
 officeWorkerSpawnInterval = **when** {  
 timeSys.gameUptime < 20 -> nextFloat(3f, 5f)  
 timeSys.gameUptime < 40 -> nextFloat(2f, 4f)  
 timeSys.gameUptime < 60 -> nextFloat(1f, 3f)  
 timeSys.gameUptime < 80 -> nextFloat(0.5f, 2.5f)  
 timeSys.gameUptime < 100 -> nextFloat(0.4f, 1.5f)  
 **else** -> nextFloat(0.3f, 1f)  
 }  
 }  
 }  
  
 **private fun** spawnOfficeWorker() {  
 **val** transformCompX = **if** (nextBoolean()) playerTransformComp.x + WORLD\_WIDTH  
 **else** playerTransformComp.x - WORLD\_WIDTH  
 **val** transformCompY = (**if** (nextBoolean()) nextInt(0, 11)  
 **else** nextInt(90, 101)).toFloat()  
  
 engine.entity **{** with<InfoComp> **{** name = **"OfficeWorker"  
 }** with<TransformComp> **{** x = transformCompX  
 y = transformCompY  
 setSizeFromTexture(textures.office\_worker\_idle)  
 **}** with<GraphicsComp>()  
 with<AnimationComp> **{** animationLoops = listOf(  
 AnimationComp.AnimationLoop(0.154f, listOf(textures.office\_worker\_run0, textures.office\_worker\_run1,  
 textures.office\_worker\_run2, textures.office\_worker\_run3, textures.office\_worker\_run4,  
 textures.office\_worker\_run5, textures.office\_worker\_run6, textures.office\_worker\_run7)),  
 AnimationComp.AnimationLoop(0.077f, listOf(textures.office\_worker\_run0, textures.office\_worker\_run1,  
 textures.office\_worker\_run2, textures.office\_worker\_run3, textures.office\_worker\_run4,  
 textures.office\_worker\_run5, textures.office\_worker\_run6, textures.office\_worker\_run7)),  
 AnimationComp.AnimationLoop(-1f, listOf(textures.office\_worker\_push))  
 )  
 determineAnimationLoop = **{** \_, action **->  
 when** (action) {  
 Action.IDLE -> 0  
 Action.RUN -> 1  
 Action.PUSH -> 2  
 **else** -> 0  
 }  
 **}** animationLoop = 2 *//set animation loop so it changes, triggering the ActionSys to record the time* **}** with<ActionComp> **{** runSpeed = 80f  
 walkSpeed = 40f  
 facing = **if** (nextBoolean()) Facing.LEFT **else** Facing.RIGHT  
 **}** with<AIComp> **{** *//States: 0 Idle  
 // 1 Run towards player  
 // 2 Attack player* determineState = **{** enemy, player **->  
 val** enemyHitbox = enemy.getNotNull(HitboxComp.mapper)  
 **val** playerHitbox = player.getNotNull(HitboxComp.mapper)  
  
 **when** {  
 *//if on different level do not attack player* abs(enemyHitbox.y - playerHitbox.y) > 50 -> 0  
  
 enemyHitbox.overlaps(playerHitbox) -> 2  
  
 **else** -> 1  
 }  
 **}** determineAction = **{** \_, \_, state **->  
 when** (state) {  
 0 -> Action.WALK  
 1 -> Action.RUN  
 2 -> Action.PUSH  
 **else** -> Action.IDLE  
 }  
 **}** determineFacing = **{** enemy, player, state **->  
 val** playerTransformComp = player.getNotNull(TransformComp.mapper)  
 **val** enemyTransformComp = enemy.getNotNull(TransformComp.mapper)  
 **val** enemyActionComp = enemy.getNotNull(ActionComp.mapper)  
  
 **when** {  
 state == 0 -> enemyActionComp.facing  
 playerTransformComp.x < enemyTransformComp.x -> Facing.LEFT  
 **else** -> Facing.RIGHT  
 }  
 **}  
 }** with<HitboxComp> **{** x = transformCompX + 9f  
 y = transformCompY  
 width = 18f  
 height = 56f  
 **}** with<HPComp> **{** deleteWhenHP0 = **true** hp = 50f  
 **}  
 }** }  
  
 **private fun** spawnOfficeWorkerKB() {  
 **val** transformCompX = **if** (nextBoolean()) playerTransformComp.x + WORLD\_WIDTH  
 **else** playerTransformComp.x - WORLD\_WIDTH  
 **val** transformCompY = (**if** (nextBoolean()) nextInt(0, 11)  
 **else** nextInt(90, 101)).toFloat()  
  
 engine.entity **{** with<InfoComp> **{** name = **"OfficeWorkerKB"  
 }** with<TransformComp> **{** x = transformCompX  
 y = transformCompY  
 setSizeFromTexture(textures.office\_worker\_idle\_kb)  
 **}** with<GraphicsComp>()  
 with<AnimationComp> **{** animationLoops = listOf(  
 AnimationComp.AnimationLoop(0.154f, listOf(textures.office\_worker\_run\_kb0,  
 textures.office\_worker\_run\_kb1, textures.office\_worker\_run\_kb2, textures.office\_worker\_run\_kb3,  
 textures.office\_worker\_run\_kb4, textures.office\_worker\_run\_kb5, textures.office\_worker\_run\_kb6,  
 textures.office\_worker\_run\_kb7)),  
 AnimationComp.AnimationLoop(0.077f, listOf(textures.office\_worker\_run\_kb0,  
 textures.office\_worker\_run\_kb1, textures.office\_worker\_run\_kb2, textures.office\_worker\_run\_kb3,  
 textures.office\_worker\_run\_kb4, textures.office\_worker\_run\_kb5, textures.office\_worker\_run\_kb6,  
 textures.office\_worker\_run\_kb7)),  
 AnimationComp.AnimationLoop(0.462f, listOf(textures.office\_worker\_hit\_kb0,  
 textures.office\_worker\_hit\_kb1))  
 )  
 determineAnimationLoop = **{** \_, action **->  
 when** (action) {  
 Action.IDLE -> 0  
 Action.RUN -> 1  
 Action.HIT\_KB -> 2  
 **else** -> 0  
 }  
 **}** animationLoop = 2 *//set animation loop so it changes, triggering the ActionSys to record the time* **}** with<ActionComp> **{** runSpeed = 80f  
 walkSpeed = 40f  
 facing = **if** (nextBoolean()) Facing.LEFT **else** Facing.RIGHT  
 **}** with<AIComp> **{** *//States: 0 Idle  
 // 1 Run towards player  
 // 2 Attack player* determineState = **{** enemy, player **->  
 val** enemyHitbox = enemy.getNotNull(HitboxComp.mapper)  
 **val** playerHitbox = player.getNotNull(HitboxComp.mapper)  
  
 **when** {  
 *//if on different level do not attack player* abs(enemyHitbox.y - playerHitbox.y) > 50 -> 0  
  
 enemyHitbox.overlaps(playerHitbox) -> 2  
  
 **else** -> 1  
 }  
 **}** determineAction = **{** \_, \_, state **->  
 when** (state) {  
 0 -> Action.WALK  
 1 -> Action.RUN  
 2 -> Action.HIT\_KB  
 **else** -> Action.IDLE  
 }  
 **}** determineFacing = **{** enemy, player, state **->  
 val** playerTransformComp = player.getNotNull(TransformComp.mapper)  
 **val** enemyTransformComp = enemy.getNotNull(TransformComp.mapper)  
 **val** enemyActionComp = enemy.getNotNull(ActionComp.mapper)  
  
 **when** {  
 state == 0 -> enemyActionComp.facing  
 playerTransformComp.x < enemyTransformComp.x -> Facing.LEFT  
 **else** -> Facing.RIGHT  
 }  
 **}  
 }** with<HitboxComp> **{** x = transformCompX + 9f  
 y = transformCompY  
 width = 18f  
 height = 56f  
 **}** with<HPComp> **{** deleteWhenHP0 = **true** hp = 50f  
 **}  
 }** }  
}

### SpriteRenderSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.SortedIteratingSystem  
**import** com.badlogic.gdx.graphics.g2d.Batch  
**import** com.badlogic.gdx.graphics.g2d.Sprite  
**import** com.badlogic.gdx.utils.viewport.Viewport  
**import** io.github.rolodophone.comboking.ckLogger  
**import** io.github.rolodophone.comboking.comp.GraphicsComp  
**import** io.github.rolodophone.comboking.comp.TransformComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.allOf  
**import** ktx.graphics.use  
  
**private val** log = ckLogger<SpriteRenderSys>()  
  
*/\*\*  
 \* Renders the entities on the screen  
 \*/***class** SpriteRenderSys(  
 **private val** batch: Batch,  
 **private val** viewport: Viewport  
): SortedIteratingSystem(  
 allOf(TransformComp::**class**, GraphicsComp::**class**).get(),  
 EntityRenderingComparator(),  
 30  
) {  
 **private val** sprite = Sprite() *//reused for each entity to draw using a batch* **override fun** update(deltaTime: Float) {  
 viewport.apply()  
 batch.use(viewport.camera.combined) **{  
 super**.update(deltaTime)  
 **}** }  
  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** graphicsComp = entity.getNotNull(GraphicsComp.mapper)  
  
 **if** (!graphicsComp.visible) **return  
  
 if** (graphicsComp.textureRegion == **null**) {  
 log.error **{ "Entity $**entity **is set to be visible but has no texture." }  
 return** }  
  
 **val** transformComp = entity.getNotNull(TransformComp.mapper)  
  
 sprite.setBounds(transformComp.x, transformComp.y, transformComp.width, transformComp.height)  
 sprite.setRegion(graphicsComp.textureRegion)  
 sprite.setFlip(graphicsComp.flippedHorizontally, **false**)  
 sprite.draw(batch)  
 }  
}  
  
**class** EntityRenderingComparator: Comparator<Entity> {  
 **override fun** compare(p0: Entity, p1: Entity): Int {  
 **val** p0TransformComp = p0.getNotNull(TransformComp.mapper)  
 **val** p1TransformComp = p1.getNotNull(TransformComp.mapper)  
  
 **return when** {  
 p0TransformComp.z < p1TransformComp.z -> -1  
 p0TransformComp.z > p1TransformComp.z -> 1  
 p0TransformComp.y < p1TransformComp.y -> -1  
 p0TransformComp.y > p1TransformComp.y -> 1  
 **else** -> 0  
 }  
 }  
}

### TextRenderSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.ashley.systems.IteratingSystem  
**import** com.badlogic.gdx.graphics.OrthographicCamera  
**import** com.badlogic.gdx.graphics.g2d.Batch  
**import** com.badlogic.gdx.math.Matrix4  
**import** com.badlogic.gdx.utils.viewport.Viewport  
**import** io.github.rolodophone.comboking.asset.ComboKingFonts  
**import** io.github.rolodophone.comboking.comp.TextComp  
**import** io.github.rolodophone.comboking.comp.TransformComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** ktx.ashley.allOf  
**import** ktx.graphics.use  
  
*/\*\*  
 \* Draws the text of entities with a* ***[TextComp]****.  
 \*/***class** TextRenderSys(  
 **private val** batch: Batch,  
 **private val** viewport: Viewport,  
 **private val** fonts: ComboKingFonts  
): IteratingSystem(  
 allOf(TransformComp::**class**, TextComp::**class**).get(), 35  
) {  
 **private lateinit var** textProjectionMatrix: Matrix4  
  
 **override fun** addedToEngine(engine: Engine) {  
 *// save the translated projection matrix for using when drawing text, so that text isn't moved when the  
 // camera moves. Translate it to put the origin in the bottom left.* **val** zoom = (viewport.camera **as** OrthographicCamera).zoom  
 textProjectionMatrix = viewport.camera.projection.cpy()  
 .translate(-viewport.worldWidth / 2f \* zoom, -viewport.worldHeight / 2f \* zoom, 0f)  
  
 **super**.addedToEngine(engine)  
 }  
  
 **override fun** update(deltaTime: Float) {  
 *// draw text using just projection matrix so that coordinates are interpreted as screen coordinates,  
 // not game world coordinates.* batch.use(textProjectionMatrix) **{  
 super**.update(deltaTime)  
 **}** }  
  
 **override fun** processEntity(entity: Entity, deltaTime: Float) {  
 **val** transformComp = entity.getNotNull(TransformComp.mapper)  
 **val** textComp = entity.getNotNull(TextComp.mapper)  
  
 fonts.visitor.color = textComp.colour  
  
 *// add the height of the font because by default text is drawn with the origin at the top left. Adding the  
 // on the height means the text is drawn with the origin in the bottom left, consistent with the rest of my game  
  
 // convert text to uppercase because this font uses the lowercase letters as slight variants of the capital  
 // letters. So I'm converting all text for the letters to look consistent* fonts.visitor.draw(batch, textComp.text.uppercase(), transformComp.x, transformComp.y + TextComp.FONT\_HEIGHT)  
 }  
}

### TimeSys

**package** io.github.rolodophone.comboking.sys  
  
**import** com.badlogic.ashley.core.EntitySystem  
  
*/\*\*  
 \* Keeps track of the app uptime and game uptime to enable timing of various things.  
 \*/***class** TimeSys : EntitySystem(-100) {  
  
 */\*\*  
 \* The number of seconds for which the app has running.  
 \*  
 \* On PC this continues increasing when the game is minimised, whereas on Android it only increases when the game  
 \* is in the foreground. On both platforms, it will never increase by more than  
 \** ***[io.github.rolodophone.comboking.screen.MAX\_DELTA\_TIME]****, so that the game state doesn't change by a huge  
 \* amount after a lag spike, and also so that the game behaves properly when being debugged.  
 \*/* **var** appUptime = 0f  
 **private set** */\*\*  
 \* The number of seconds for which the game has been running, or -1 if the game isn't running  
 \*  
 \* On PC this continues increasing when the game is minimised, whereas on Android it only increases when the game  
 \* is in the foreground. On both platforms, it will never increase by more than  
 \** ***[io.github.rolodophone.comboking.screen.MAX\_DELTA\_TIME]****, so that the game state doesn't change by a huge  
 \* amount after a lag spike, and also so that the game behaves properly when being debugged.  
 \*/* **var** gameUptime = -1f  
  
 **override fun** update(deltaTime: Float) {  
 appUptime += deltaTime  
  
 *//increment gameUptime if game is running* **if** (gameUptime != -1f) gameUptime += deltaTime  
 }  
}

## core: util

### Util

@**file**:Suppress(**"unused"**)  
  
**package** io.github.rolodophone.comboking.util  
  
**import** com.badlogic.ashley.core.Component  
**import** com.badlogic.ashley.core.ComponentMapper  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.gdx.math.Rectangle  
**import** com.badlogic.gdx.math.Vector2  
**import** com.badlogic.gdx.utils.viewport.Viewport  
**import** io.github.rolodophone.comboking.comp.InfoComp  
**import** ktx.ashley.get  
**import** kotlin.random.Random.Default.nextFloat  
  
**private val** tempVector = Vector2()  
  
*/\*\*  
 \* Transforms the specified screen x coordinate to world coordinates.  
 \*/***fun** Viewport.unprojectX(x: Float): Float {  
 tempVector.x = x  
 **return** unproject(tempVector).x  
}  
  
*/\*\*  
 \* Transforms the specified y screen coordinate to world coordinates.  
 \*/***fun** Viewport.unprojectY(y: Float): Float {  
 tempVector.y = y  
 **return** unproject(tempVector).y  
}  
  
**fun** Viewport.halfWorldWidth() = worldWidth / 2f  
**fun** Viewport.halfWorldHeight() = worldHeight / 2f  
  
**fun** Rectangle.halfWidth() = width / 2  
**fun** Rectangle.halfHeight() = height / 2  
  
**operator fun** Vector2.plus(vec: Vector2): Vector2 = **this**.add(vec)  
  
**operator fun** Vector2.times(scalar: Float): Vector2 = **this**.scl(scalar)  
**operator fun** Float.times(vec: Vector2) = vec \* **this***/\*\*  
 \* Get the specified* ***[Component]*** *from an* ***[Entity]****, throwing an error if the* ***[Component]*** *isn't found.  
 \*/***fun** <T : Component> Entity.getNotNull(mapper: ComponentMapper<T>): T {  
 **val** component = **this**[mapper]  
 requireNotNull(component) **{ "${**repr()**} hasn't got the requested component" }  
 return** component  
}  
  
**fun** nextFloat(from: Float, until: Float) = (nextFloat() \* (until - from)) + from  
  
*// string representations for logging***fun** Entity.repr() = **"${**get(InfoComp.mapper)?.name ?: **"Unnamed"}@${**hashCode()**}"**

## core: top-level

### CKLogger

**package** io.github.rolodophone.comboking  
  
**import** ktx.log.Logger  
  
*/\*\*  
 \* Wrapper for libKTX* ***[Logger]*** *that logs only the tag and the message, without logging the debug level.  
 \*/***class** CKLogger(tag: String): Logger(**""**, tag, tag, tag) {  
 **override fun** buildMessage(message: String) = message  
}  
  
**inline fun** <**reified** T : Any> ckLogger(): Logger = CKLogger(T::**class**.simpleName ?: **"Anonymous"**)

### CKPrefs

**package** io.github.rolodophone.comboking  
  
**import** com.badlogic.gdx.Gdx  
  
*/\*\*  
 \* Manages putting values to and getting values from the persistent storage.  
 \*/***class** CKPrefs {  
 **private val** prefs = Gdx.app.getPreferences(**"io.github.rolodophone.comboking"**)  
  
 **fun** putSFXVolume(value: Float) {  
 prefs.putFloat(**"sfx\_volume"**, value)  
 prefs.flush()  
 }  
  
 **fun** putMusicVolume(value: Float) {  
 prefs.putFloat(**"music\_volume"**, value)  
 prefs.flush()  
 }  
  
 **fun** putHighscore(value: Int) {  
 prefs.putInteger(**"highscore"**, value)  
 prefs.flush()  
 }  
  
 **fun** getSFXVolume() = prefs.getFloat(**"sfx\_volume"**, 1f)  
  
 **fun** getMusicVolume() = prefs.getFloat(**"music\_volume"**, 1f)  
  
 **fun** getHighscore() = prefs.getInteger(**"highscore"**, 0)  
}

### ComboKing

**package** io.github.rolodophone.comboking  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.PooledEngine  
**import** com.badlogic.gdx.Application.LOG\_DEBUG  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.InputMultiplexer  
**import** com.badlogic.gdx.InputProcessor  
**import** com.badlogic.gdx.graphics.g2d.Batch  
**import** com.badlogic.gdx.graphics.g2d.SpriteBatch  
**import** com.badlogic.gdx.utils.viewport.FitViewport  
**import** io.github.rolodophone.comboking.asset.ComboKingFonts  
**import** io.github.rolodophone.comboking.asset.ComboKingMusic  
**import** io.github.rolodophone.comboking.asset.ComboKingSounds  
**import** io.github.rolodophone.comboking.asset.ComboKingTextures  
**import** io.github.rolodophone.comboking.event.GameEventManager  
**import** io.github.rolodophone.comboking.screen.ComboKingScreen  
**import** io.github.rolodophone.comboking.screen.GameOverScreen  
**import** io.github.rolodophone.comboking.screen.GameScreen  
**import** io.github.rolodophone.comboking.screen.MenuScreen  
**import** io.github.rolodophone.comboking.sys.MusicSys  
**import** io.github.rolodophone.comboking.sys.SpriteRenderSys  
**import** io.github.rolodophone.comboking.sys.TimeSys  
**import** ktx.app.KtxGame  
  
*// 16x9 aspect ratio with a highest common factor of 20. This means I can scale by any multiple of 1/20***const val** WORLD\_WIDTH = 16\*20  
**const val** WORLD\_HEIGHT = 9\*20  
  
**private const val** BATCH\_SIZE = 1000  
  
**private val** log = ckLogger<ComboKing>()  
  
  
*/\*\*  
 \* The main class. This is created from a platform-specific launcher to start the app.  
 \*  
 \** ***@param createPlayerInputProcessor*** *The function that creates the* ***[InputProcessor]*** *that will be used to detect the  
 \* game controls input. This is passed in as a parameter as the controls are different on Android and PC.  
 \*/***class** ComboKing(  
 **private val** createPlayerInputProcessor: (GameEventManager: GameEventManager, screenWidth: Int, screenHeight: Int)  
 -> InputProcessor  
): KtxGame<ComboKingScreen>() {  
  
 **val** viewport = FitViewport(WORLD\_WIDTH.toFloat(), WORLD\_HEIGHT.toFloat())  
 **val** gameEventManager = GameEventManager()  
 **lateinit var** batch: Batch  
 **lateinit var** comboKingTextures: ComboKingTextures  
 **lateinit var** comboKingFonts: ComboKingFonts  
 **lateinit var** comboKingMusic: ComboKingMusic  
 **lateinit var** comboKingSounds: ComboKingSounds  
 **lateinit var** ckPrefs: CKPrefs  
 **lateinit var** engine: Engine  
  
 **lateinit var** timeSys: TimeSys  
  
 **override fun** create() {  
 Gdx.app.logLevel = LOG\_DEBUG  
  
 *//init stuff* batch = SpriteBatch(BATCH\_SIZE)  
 comboKingTextures = ComboKingTextures()  
 comboKingFonts = ComboKingFonts()  
 comboKingMusic = ComboKingMusic()  
 comboKingSounds = ComboKingSounds()  
 ckPrefs = CKPrefs()  
 engine = PooledEngine()  
  
 Gdx.input.inputProcessor = InputMultiplexer(UIInputProcessor(engine, viewport))  
  
 addScreen(MenuScreen(**this**))  
 addScreen(GameScreen(**this**, createPlayerInputProcessor))  
 addScreen(GameOverScreen(**this**))  
  
 timeSys = TimeSys()  
  
 *//add systems to engine* engine.run **{** addSystem(timeSys)  
 addSystem(SpriteRenderSys(batch, viewport))  
 addSystem(MusicSys(comboKingMusic, **this**@ComboKing, ckPrefs))  
 **}** comboKingSounds.setVolume(ckPrefs.getSFXVolume())  
  
 setScreen<MenuScreen>()  
 }  
  
 **override fun** dispose() {  
 log.debug **{ "Disposing game" }  
  
 super**.dispose() *// disposes all registered screens* log.debug **{  
 val** sb = batch **as** SpriteBatch  
 **"Max sprites in batch: ${**sb.maxSpritesInBatch**}; size of batch: $**BATCH\_SIZE**"  
 }** batch.dispose()  
  
 comboKingTextures.dispose()  
 comboKingFonts.dispose()  
 comboKingMusic.dispose()  
 comboKingSounds.dispose()  
 }  
}

### UIInputProcessor

**package** io.github.rolodophone.comboking  
  
**import** com.badlogic.ashley.core.Engine  
**import** com.badlogic.ashley.core.Entity  
**import** com.badlogic.gdx.Gdx  
**import** com.badlogic.gdx.Input  
**import** com.badlogic.gdx.InputProcessor  
**import** com.badlogic.gdx.utils.viewport.Viewport  
**import** io.github.rolodophone.comboking.comp.ButtonComp  
**import** io.github.rolodophone.comboking.comp.GraphicsComp  
**import** io.github.rolodophone.comboking.comp.SliderComp  
**import** io.github.rolodophone.comboking.comp.TransformComp  
**import** io.github.rolodophone.comboking.util.getNotNull  
**import** io.github.rolodophone.comboking.util.unprojectX  
**import** io.github.rolodophone.comboking.util.unprojectY  
**import** ktx.ashley.allOf  
**import** ktx.ashley.contains  
**import** ktx.ashley.get  
**import** ktx.ashley.oneOf  
  
*/\*\*  
 \* Handles touch and key events for UI elements.  
 \*/***class** UIInputProcessor(  
 **private val** engine: Engine,  
 **private val** viewport: Viewport  
): InputProcessor {  
 *// Each method returns true to indicate that the event shouldn't be passed on to other input processors or false   
 // to indicate that it should be passed on.* **private var** elementPressed: Entity? = **null  
  
 override fun** keyDown(keycode: Int): Boolean {  
 *//toggle borderless fullscreen when F11 pressed* **if** (keycode == Input.Keys.F11) {  
 **if** (Gdx.graphics.isFullscreen) {  
 Gdx.graphics.setWindowedMode(WORLD\_WIDTH\*3, WORLD\_HEIGHT\*3)  
 }  
 **else** {  
 Gdx.graphics.setFullscreenMode(Gdx.graphics.displayMode)  
 }  
 }  
  
 **return false** }  
  
 **override fun** keyUp(keycode: Int): Boolean {  
 **return false** }  
  
 **override fun** keyTyped(character: Char): Boolean {  
 **return false** }  
  
 **override fun** touchDown(screenX: Int, screenY: Int, pointer: Int, button: Int): Boolean {  
 *// check if a button was pressed down* **val** elements = engine.getEntitiesFor(  
 allOf(TransformComp::**class**)  
 .oneOf(ButtonComp::**class**, SliderComp::**class**)  
 .get())  
  
 **for** (element **in** elements) {  
 **val** transformComp = element.getNotNull(TransformComp.mapper)  
  
 **if** (transformComp.contains(  
 viewport.unprojectX(screenX.toFloat()),  
 viewport.unprojectY(screenY.toFloat())  
 )) {  
 elementPressed = element  
 **return true** }  
 }  
  
 **return false** }  
  
 **override fun** touchUp(screenX: Int, screenY: Int, pointer: Int, button: Int): Boolean {  
 elementPressed?.let **{** element **->  
 if** (element.contains(ButtonComp.mapper)) { *//only concerned about touch up events on buttons, not sliders  
  
 // check if the button that was pressed was released* **val** transformComp = element.getNotNull(TransformComp.mapper)  
  
 **if** (transformComp.contains(  
 viewport.unprojectX(screenX.toFloat()),  
 viewport.unprojectY(screenY.toFloat())  
 )  
 ) {  
 *//execute button action* **val** buttonComp = element.getNotNull(ButtonComp.mapper)  
 buttonComp.onPress()  
 elementPressed = **null  
 return true** }  
 }  
 **}** elementPressed = **null  
 return false** }  
  
 **override fun** touchDragged(screenX: Int, screenY: Int, pointer: Int): Boolean {  
 elementPressed?.let **{** element **->  
 if** (element.contains(SliderComp.mapper)) { *//only concerned about touch drag events on sliders, not buttons* **val** transformComp = element.getNotNull(TransformComp.mapper)  
 **val** graphicsComp = element[GraphicsComp.mapper]  
 **val** sliderComp = element.getNotNull(SliderComp.mapper)  
 **val** worldX = viewport.unprojectX(screenX.toFloat())  
  
 **when** {  
 worldX < transformComp.left + transformComp.width \* (1/8f) -> {  
 **if** (sliderComp.progress != 0f) {  
 sliderComp.progress = 0f  
 graphicsComp?.textureRegion = sliderComp.textures[0]  
 sliderComp.onChange(sliderComp.progress)  
 }  
 }  
 worldX < transformComp.left + transformComp.width \* (3/8f) -> {  
 **if** (sliderComp.progress != 0.25f) {  
 sliderComp.progress = 0.25f  
 graphicsComp?.textureRegion = sliderComp.textures[1]  
 sliderComp.onChange(sliderComp.progress)  
 }  
 }  
 worldX < transformComp.left + transformComp.width \* (5/8f) -> {  
 **if** (sliderComp.progress != 0.5f) {  
 sliderComp.progress = 0.5f  
 graphicsComp?.textureRegion = sliderComp.textures[2]  
 sliderComp.onChange(sliderComp.progress)  
 }  
 }  
 worldX < transformComp.left + transformComp.width \* (7/8f) -> {  
 **if** (sliderComp.progress != 0.75f) {  
 sliderComp.progress = 0.75f  
 graphicsComp?.textureRegion = sliderComp.textures[3]  
 sliderComp.onChange(sliderComp.progress)  
 }  
 }  
 **else** -> {  
 **if** (sliderComp.progress != 1f) {  
 sliderComp.progress = 1f  
 graphicsComp?.textureRegion = sliderComp.textures[4]  
 sliderComp.onChange(sliderComp.progress)  
 }  
 }  
 }  
 }  
  
 **return true  
 }  
  
 return false** }  
  
 **override fun** mouseMoved(screenX: Int, screenY: Int): Boolean {  
 **return false** }  
  
 **override fun** scrolled(amountX: Float, amountY: Float): Boolean {  
 **return false** }  
}

# Appendix

## Initial Communication with Third Party

**What do you think makes a good endless running game?**

* High score definitely
  + Was already going to do this
* Should change each time you play it
  + Will be randomised, and I will implement a leveling up system
* Needs some level of randomisation
  + The layout and enemies will be randomised
* Pickups and items
  + I will implement a small number of items

**Can you give some examples of items or power-ups that I can include in my game?**

* Health pickup, like bandages
  + I will implement bandages
* Different weapons that you can pick up
  + I will implement a pistol but that will likely be the only weapon, and Jane will start out with it

**What important thing do many games get wrong?**

* Become stale quickly
  + I will implement leveling up, and the layout will be randomised so no two runs will be the same
* If there's no progress, you're going to get bored quickly
  + You will be able to level up, which should create a sense of progress
* Needs a highscore or leveling up
  + See above
* Leveling up means permanently improving the player's abilities (e.g. more health) using XP earned by getting far
  + This is how leveling up will work in my game

**Other ideas that came up in the conversation**

* Swipe with knife for area of effect damage
  + I will not implement a knife as that would make punching and kicking useless
* Buttons to change between weapons you have picked up
  + Instead of another button, the user will be able to draw a circle with their right thumb to switch between using fists and using the pistol
* There could be a pistol but with limited ammunition, which could be picked up as a rare item
  + I will do this, and the pistol's ammo capacity can be another ability that you can level up

## Feedback from Third Party

TODO

# References

[1] <https://www.levelwinner.com/prizefighters-2-guide-13-tips-tricks-strategies-to-win-more-fights-and-become-a-champion/>

[2] <https://en.wikipedia.org/wiki/Bushnell%27s_Law>

[3] <https://play.google.com/store/apps/details?id=com.nekki.shadowfight&hl=en_GB&gl=US>

[4] [https://play.google.com/store/apps/details?id=com.koalitygame.prizefighters2&hl=en\_GB&gl= US](https://play.google.com/store/apps/details?id=com.koalitygame.prizefighters2&hl=en_GB&gl=%20US)

[5] <https://gameprogrammingpatterns.com/contents.html>

[6] <https://gameprogrammingpatterns.com/architecture-performance-and-games.html#:~:text=arc hitecture%20is%20about%20change>

[7] <https://gameprogrammingpatterns.com/introduction.html#:~:text=that%20isn%E2%80%99t%2 0to%20imply%20that%20these%20patterns%20are%20only%20useful%20in%20that%20language>

[8] <https://gameprogrammingpatterns.com/architecture-performance-and-games.html#:~:text=per formance%20is%20all%20about%20assumptions>

[9] <https://libgdx.com/>

[10] <https://kotlinlang.org/>

[11] <https://libktx.github.io/>

[12] <https://box2d.org/>

[13] <https://github.com/libgdx/libgdx/wiki/Scene2d>