UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

ENG1 Assessment 1
Group 18 - Octodecimal

Requirements

Group Members:

Izz Abd Aziz

Phil Suwanpimolkul

Tom Loomes

Owen Kilpatrick

Zachary Vickers

Michael Ballantyne

Architecture Design Process

Our team used Responsibility-Driven Design (RDD) as the method to create the initial design of our system. It is specialised for object-oriented design which is how we have decided to implement the product. The aim of RDD is to maximise abstraction, distribute behaviour and provide flexibility [1].

The first step was to consider the product brief, interview with the client and requirements for a detailed description of the system. A designer story was developed to help us understand the key parts of the design. By underlining nouns in the product brief the main candidate objects were found based on the themes. With these we created CRC (Candidate, Responsibilities, Collaborators) cards, each with a small description of the concept and stereotypes. Next, from grouping the CRC cards it was clear some were unnecessary as they duplicated functionality so were removed. For instance, the Cell card was unnecessary as the player can move freely throughout the map so it doesn't need to be split into squares. Also, the GamePauser is not required as this can be done in GameScreen. Finally, individual responsibilities and collaborators were added to the cards. Collaborators are other cards that will need to be interacted with in order to meet responsibilities. These initial CRC cards with responsibilities and collaborators can be seen on the website [https://eng1team18.github.io/website-2/crc_cards.html].

Creating CRC cards is merely an initial estimate of what classes will be required to fulfil the product brief. When we were happy after looking through this a few times, we moved onto trying to map out these CRC cards to UML diagrams. At the outset we started with sketches drawn by hand as this allows for informal discussion where we don't have to focus on syntax and just lay out ideas. Then we moved to a tool called plantUML for formal UML diagrams from the sketches. A variety of diagrams were made to show the structure and behaviour of the system including class, sequence and state diagrams. Many iterations of each diagram were created throughout the project as new features and improvements were made.

UML Diagrams

Tools used

To create the structural and behavioural diagrams needed to represent the system we used plantUML. One reason we selected it was because it can be used across multiple different types of platforms: in browser; embedded in a Google Document with the plantUML Gizmo extension and with IntelliJ IDEA's plugin by simply making a .puml file. As we are already using IntelliJ for the implementation it's an IDE the whole team should already have installed and is available on lab computers. The code is very human-readable and the documentation is well developed with lots of examples making it simple to learn and implement. One issue with PlantUML is that in the diagrams the arrows can go in sub-optimal routes which can overcomplicate them. The text was also often very small so to fix these issues we tried altering the arrow length and text size.

Structural Diagrams-

Class diagram with packages for the whole system

When creating the initial class diagram https://eng1team18.github.io/website-2/architecture.html] it was clear it would be very cluttered as there are many classes so we broke it down into packages where possible. The Screen package was for all screens used throughout the game (MenuScreen, GameScreen and SettingsScreen). The Event package was for coordinating and managing all the in-game events (EventManager, Activity and OptionDialogue). GameObject and Location are in the Environment package as they are to be placed throughout the map. DateTime is a package for the date and time as they are closely linked and rely on each other when it comes to incrementing the day. HustleGame, Player, Map and Energy didn't quite fit into packages so have been left alone.

For the next version [https://eng1team18.github.io/website-2/architecture.html], an interface called Screen was added as all screens had attributes/methods in common but no Screen instance will ever need to be created. This means all screen classes in this package will inherit from the Screen class. A SettingsScreen was also added as we realised a separate screen would be best for this rather than including it in the MenuScreen. The Screen package was changed to UserInterface so as not to confuse with the new interface also called Screen. The map class was removed as in the game it would be an asset rather than its own class. Relationships between classes/package classes were changed so Environment and Event now relate to GameScreen instead of HustleGame. This is because they are only needed and will be rendered/used on this screen.

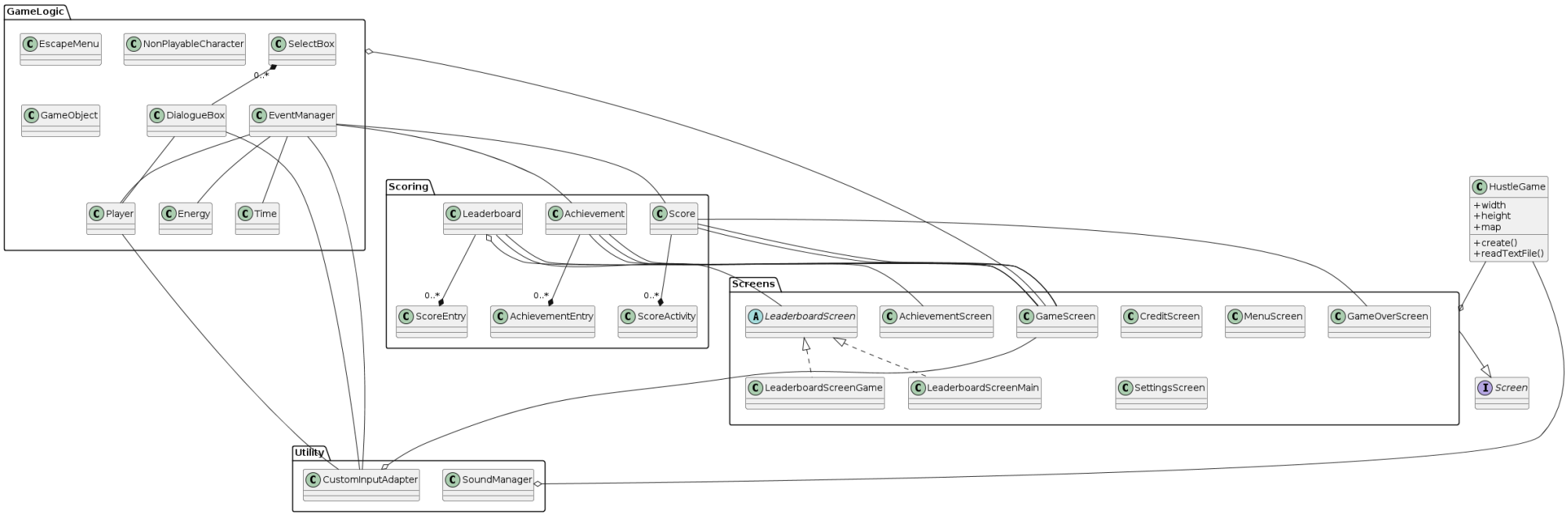
For the third version of the class diagram, a CreditScreen was added as this is now necessary, as well as setup screens as methods in MenuScreen for the tutorial and avatar selection which shouldn't need their own class. A GameOver screen was also added which implements the Screen interface. This displays final stats and has a button leading to the MenuScreen. Music and sound effects were not necessary but we had time to implement them and thought they would be a nice addition so a SoundManager class was created to control how sounds are used in the game. OptionDialogue was also renamed to DialogueBox as it was deemed a clearer name.

This third class diagram also has additional images of each package expanded, which can be seen on the website. There is one event manager but only one instance. There can be many activities for the event manager to coordinate. In Environment, Location inherits from GameObject as it will use the same methods but needs more to track what type of location it is and how many times it has been visited. In UserInterface - MenuScreen GameScreen, SettingsScreen, CreditScreen and GameOverScreen all implement the Screen interface as this has methods all will use but will not be created.

New Stuff for class diagram

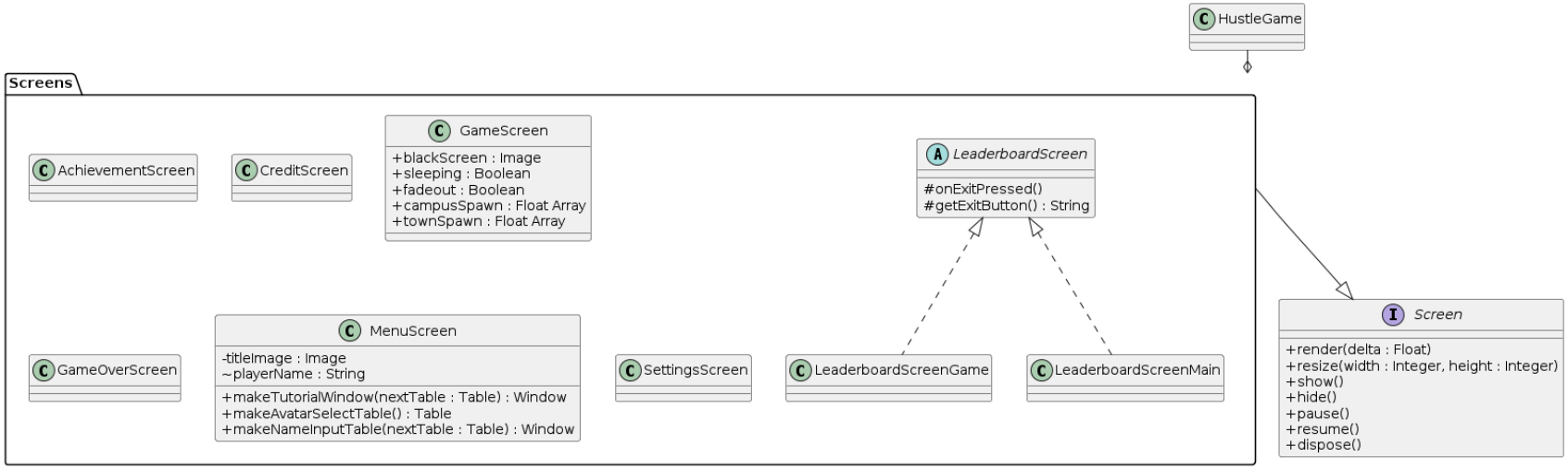
Our class diagram was finally updated again during the second assessment, the final version can be seen below. The new brief led to changes in our requirements, hence new classes were needed to implement these new features in our architecture. The new screens needed were a LeaderboardScreen to display a list of the highest scores (which has two variants for access from different screens) and an AchievementScreen to show the player their achievements (streaks) at the end of the game. The second assessment brief also required the addition of a scoring system, as well as a leaderboard and achievements as aforementioned. This led to us adding Score, Achievement and Leaderboard classes to perform the logic of these functionalities, as well as each of these having composition over one private class to store one entry/activity. After asking feedback from our client, we discovered the game needs NPCs to represent other students, so an NPC class was also added to the diagram. We also moved some of the logical functionality of GameScreen into separate classes, as this refactoring of the code improved our architecture's modularity and the corresponding code's testability. This led to the addition of Time and Energy classes, as well as a CustomInputAdapter class to act on inputs during gameplay.

As can be seen above, these classes have been repackaged based on their functionality. HustleGame was left unpackaged as this is the main class that generates the game. The package "screens" contains all screen classes, "gamelogic" are all logic based classes used by GameScreen, "scoring" contains all classes directly related to our scoring system, "utility" contains all other useful classes. All 1…1 cardinalities are not shown, as this is considered the default.

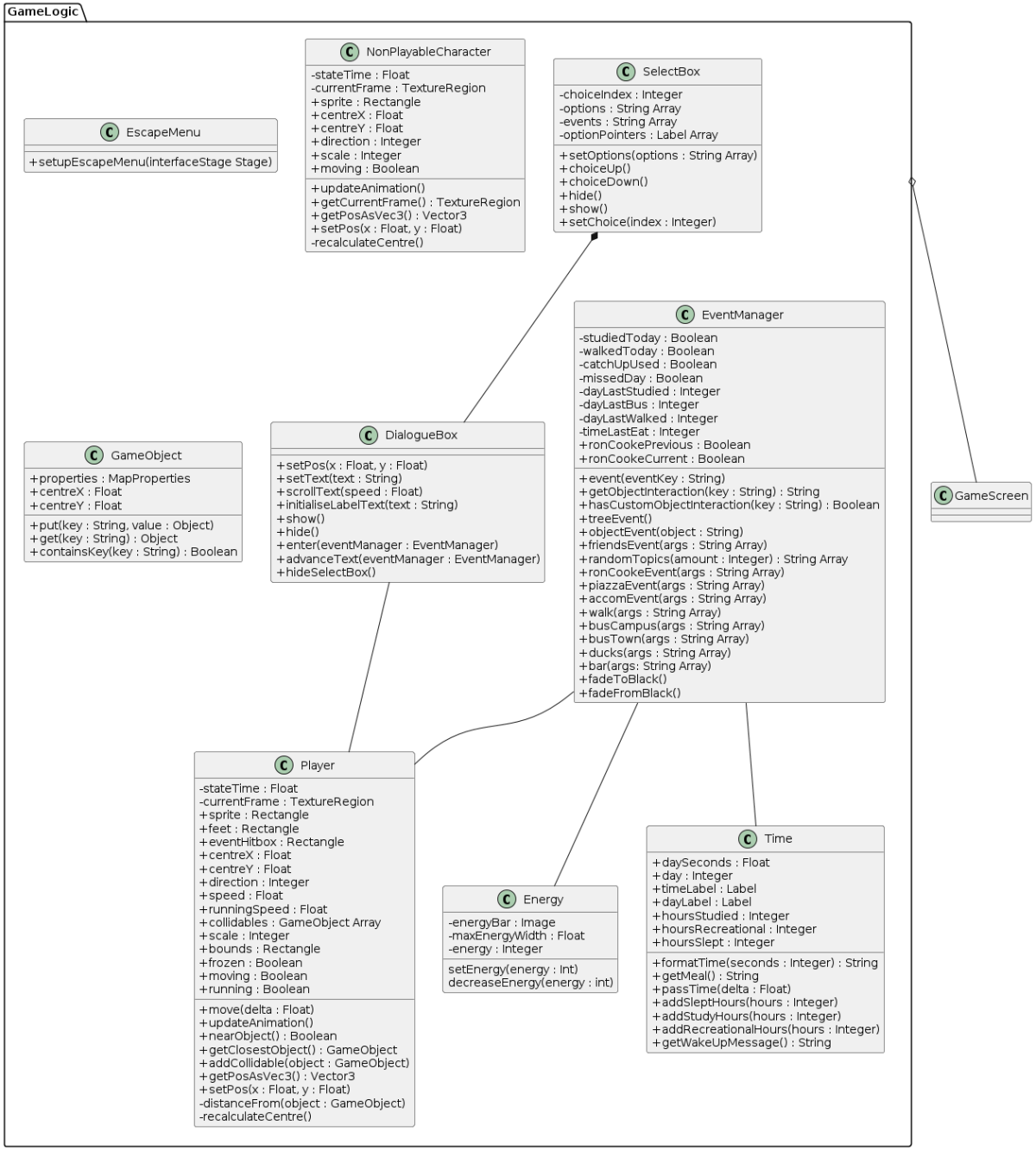


Each package has been expanded on with more information about their classes' attributes and methods (excluding some intricate implementation details). This restricts the size of the overall class diagram and increases the usability of our structural diagrams. Note that in the diagram, some relationships are connected to packages rather than individual classes, this is to increase the visibility of the diagram, it means that the relationship applies to every class in that package. Some of these relationship arrows may appear to be floating, as PlantUML may not always connect the line directly to the package rectangle.

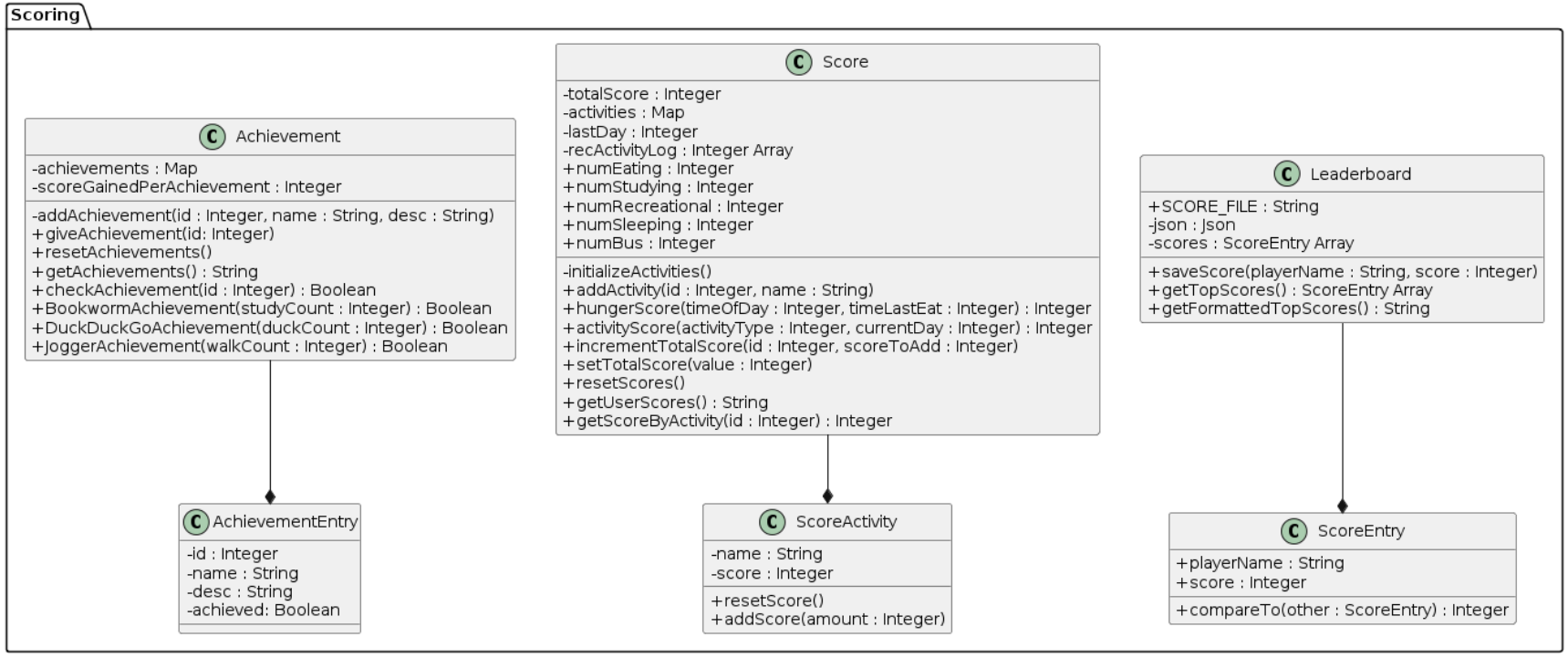
Below are the screens, these all inherit generic screen methods from a general screen interface. In the implementation, this is provided by LibGDX, however it's still included in the architecture as our screen classes must override and change these methods. Some of the screens below don't have any of their own attributes or methods, as they only contain the overridden methods from Screen. The below image doesn't include most relationships to classes outside of the package for increased visual clarity, all their relationships as can be seen in the complete class diagram previously shown.



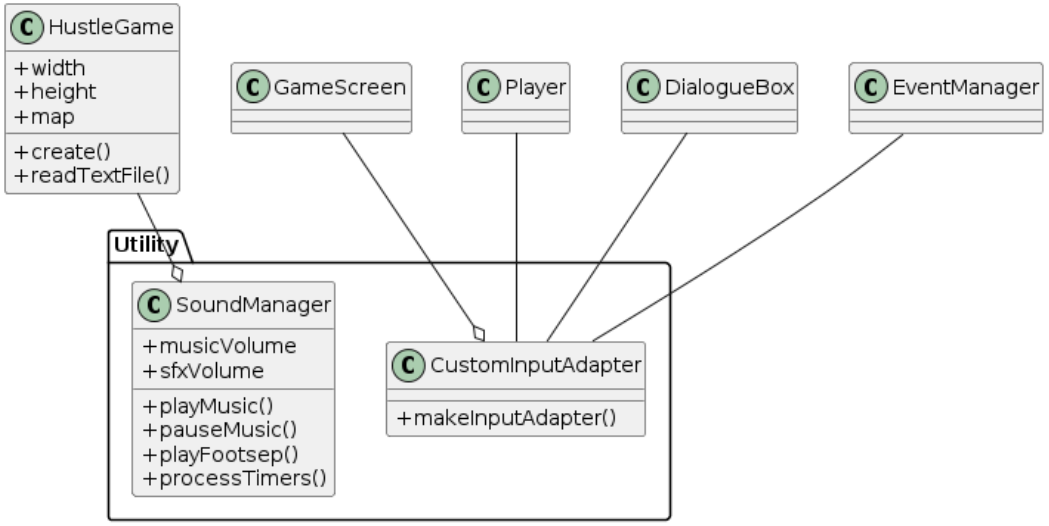
Next is the gamelogic package, GameScreen also remains in this diagram to make it clear that these classes are all instantiated in GameScreen. Their relevant attributes and methods are included, most of their relationships to classes outside of the package are removed for visual clarity.


GameLogic
NonPlayableCharacter
-stateTime : Float
-currentFrame : TextureRegion
+sprite : Rectangle
+centreX : Float
+centreY : Float
+direction : Integer
+scale : Integer
+moving : Boolean
+updateAnimation()
+getCurrentFrame() : TextureRegion
+getPosAsVec3() : Vector3
+setPos(x : Float, y : Float)
-recalculateCentre()
SelectBox
-choiceIndex : Integer
-options : String Array
-events : String Array
-optionPointers : Label Array
+setOptions(options : String Array)
+choiceUp()
+choiceDown()
+hide()
+show()
+setChoice(index : Integer)
EscapeMenu
+setupEscapeMenu(interfaceStage Stage)
EventManager
-studiedToday : Boolean
-walkedToday : Boolean
-catchUpUsed : Boolean
-missedDay : Boolean
-dayLastStudied : Integer
-dayLastBus : Integer
-dayLastWalked : Integer
-timeLastEat : Integer
+ronCookePrevious : Boolean
+ronCookeCurrent : Boolean
+event(eventKey : String)
+getObjectInteraction(key : String) : String
+hasCustomObjectInteraction(key : String) : Boolean
+treeEvent()
+objectEvent(object : String)
+friendsEvent(args : String Array)
+randomTopics(amount : Integer) : String Array
+ronCookeEvent(args : String Array)
+piazzaEvent(args : String Array)
+accomEvent(args : String Array)
+walk(args : String Array)
+busCampus(args : String Array)
+busTown(args : String Array)
+ducks(args : String Array)
+bar(args: String Array)
+fadeToBlack()
+fadeFromBlack()
DialogueBox
+setPos(x : Float, y : Float)
+setText(text : String)
+scrollText(speed : Float)
+initialiseLabelText(text : String)
+show()
+hide()
+enter(eventManager : EventManager)
+advanceText(eventManager : EventManager)
+hideSelectBox()
GameObject
+properties : MapProperties
+centreX : Float
+centreY : Float
+put(key : String, value : Object)
+get(key : String) : Object
+containsKey(key : String) : Boolean
GameScreen
Player
-stateTime : Float
-currentFrame : TextureRegion
+sprite : Rectangle
+feet : Rectangle
+eventHitbox : Rectangle
+centreX : Float
+centreY : Float
+direction : Integer
+speed : Float
+runningSpeed : Float
+collidables : GameObject Array
+scale : Integer
+bounds : Rectangle
+frozen : Boolean
+moving : Boolean
+running : Boolean
+move(delta : Float)
+updateAnimation()
+nearObject() : Boolean
+getClosestObject() : GameObject
+addCollidable(object : GameObject)
+getPosAsVec3() : Vector3
+setPos(x : Float, y : Float)
-distanceFrom(object : GameObject)
-recalculateCentre()
Energy
-energyBar : Image
-maxEnergyWidth : Float
-energy : Integer
setEnergy(energy : Int)
decreaseEnergy(energy : int)
Time
+daySeconds : Float
+day : Integer
+timeLabel : Label
+dayLabel : Label
+hoursStudied : Integer
+hoursRecreational : Integer
+hoursSlept : Integer
+formatTime(seconds : Integer) : String
+getMeal() : String
+passTime(delta : Float)
+addSleptHours(hours : Integer)
+addStudyHours(hours : Integer)
+addRecreationalHours(hours : Integer)
+getWakeUpMessage() : String

Below is the scoring package, this includes all the logical classes directly related to scoring (including achievements and leaderboard). As with the previous package diagrams, the relevant methods and attributes are included, but relationships with most classes outside of the package are removed for visual clarity.



Finally, the utility package with more information including attributes and methods is below. This diagram didn't have many relationships outside of the package, and the utility package is small, so the relationships outside of the package were all left in the diagram, as there is still visual clarity.



Behavioural diagrams - https://eng1team18.github.io/website-2/architecture_new.html

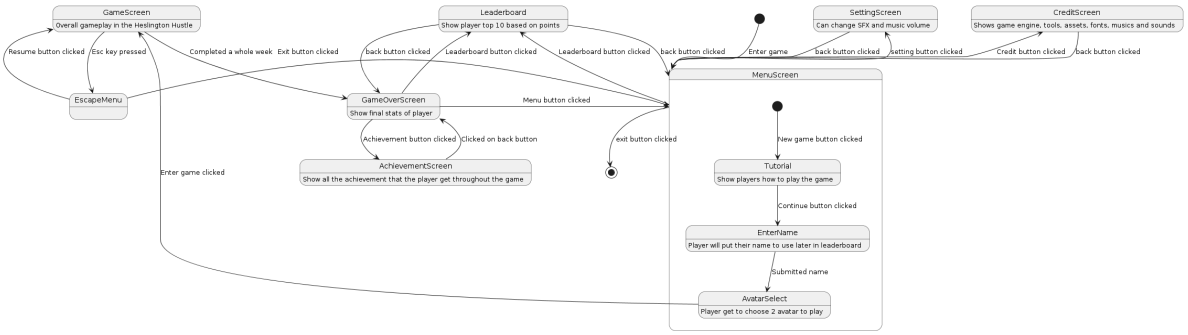
State diagram for screens

For the initial version of this state diagram from Group 16, we realised that we shared almost the same state diagram for Assessment 1. Some states like “MenuScreen”, “AvatarSelect”, “CreditScreen”, “GameScreen” and “GameOverScreen” are implemented in both state diagrams. For the first version of Assessment 2 state diagram, we added “EnterName”, “AchievementScreen” and “LeaderBoard” because of new requirements like “UR-LEADERBOARD”, “UR-USERNAME”, “UR-ACHIEVEMENT”.

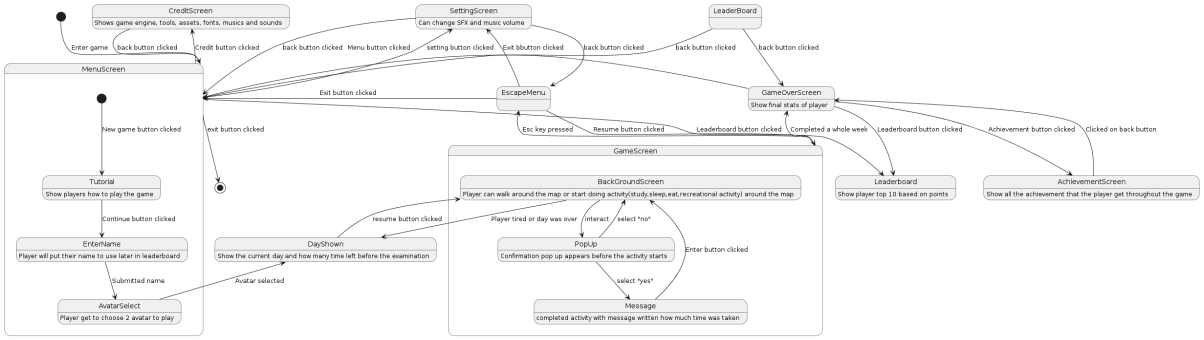
When a player enters the game, the MenuScreen will immediately appear on the screen. It matches our user requirement that the player can see the main menu screen to start and quit the game. This state gives the player 3 options to start with which is quit, start, credits, setting and leaderboard. The first iteration of our state diagram was a simple diagram that showed the loop between CreditScreen and MenuScreen, SettingScreen and MenuScreen, or Leaderboard and MenuScreen. Then, the state has the second iteration which iterates from AvatarSelect until GameOverScreen which shows overall gameplay of Heslington Hustle. Then, the player can quit or go back to the main menu.

When the player chooses to start, the player is provided a selection of avatars to choose from. This is due to our user requirement of ID UR_AVATARS. Only when the player selects the avatar they want to play as, then the game starts in a composite state called GameScreen. At this state, activities and actions are repeated until the avatar is tired or the day is over. This is also one of our user requirements which is UR_ACTIVITIES and UR_ACTIONS. At the end of the day, the player needs to sleep which is UR_DAILY_ROUTINE in our user requirements.

After completing the whole week, the system will be in the new state which is GameOverScreen. Every action, activity and routine will be calculated and shown during this state. The player can see the final score obtained when they play the game. Finally, the player can choose whether to go back to the main menu, quit the game, see leaderboard or see achievements.



For the final version of the state diagram, the composite state GameScreen should be updated to give more details of what happens inside it. States like “BackGroundScreen”, “PopUp” and ”Message” were added to show loops inside the GameScreen. This is also one of our user requirements which is UR_INTERACT and UR_WORLD. In addition, DayShown was added to give transitions between AvatarSelect and GameScreen.



Relating Architecture to Requirements

User Requirements

ID	Architecture
UR-MENU	There is a MenuScreen class which displays all the appropriate options to navigate as intended. This can also be seen at MenuScreen in the state diagram, as it shows all the different states that can be reached from the menu.
UR-CUSTOMISE	In the state diagram, there is an AvatarSelect state which represents a screen where the user can personalise their character.
UR-WORLD	The GameScreen class renders the map. The state BackgroundScreen in the state diagram represents the displaying of the map to the user.
UR-INTERACT	In the class diagram, we can see when the Player approaches a GameObject, interaction options appear as a DialogueBox. Methods for doing this are needed.
UR-TIMED	The Time class in the class diagram keeps track of the time (including the day). It is displayed using the GameScreen class.
UR-INFO	Looking at the class diagram, the Energy class stores the energy level of the Player and it is represented as a bar on the GameScreen.
UR-SOUND	The SoundManager class manages when music and sounds are played. It also controls the music volume and sfx volume.
UR-SETTINGS	The SettingsScreen class allows the user to change the music volume and sfx volume.
UR-SLEEP	The class diagram has an EventManager, this class will handle the replenishing of the character's energy levels using the Energy class when a sleeping event is triggered.
UR-LEADERBOARD	The class diagram contains Leaderboard and ScoreEntry which are used to track a leaderboard of the best scores. The LeaderboardScreen classes' subclasses are responsible for displaying this leaderboard to the user. The LeaderboardScreen can also be seen in the state diagram.
UR-ACHIEVEMENT	The Achievements class is used to keep track of achievements, these are displayed to the user in the AchievementScreen state.
UR-AVATAR	The GameObject class is used to put objects into the world. The Player class has a move() method which handles the collisions appropriately.
UR-NPCS	The NonPlayableCharacter class is used to create these NPCs into the map, its methods allow interactions.
UR-CREDITS	The state diagram has a CreditsScreen state, this is where the user would be shown the credits.
UR-USERNAME	The user can choose a name in the EnterName state of the state diagram.

UR-SCORE-BREAKDOWN	The score breakdown is calculated by the Score class. It can be viewed in the GameOverScreen state of the state diagram.

Functional System Requirements

ID	Architecture
FR-VIEW	The game uses top down graphics and 3rd person sprites with arrow keys that allows the user to move North, East, South and West according to WASD and Arrow keys
FR-NAVIGATE	In state diagram, player can navigate around map when they in the GameScreen
FR-SPRINT	In the class diagram, sprint can be seen as “running” attributes at the Player class.
FR-START	Requires the player to be able to select between avatars which is fulfilled by the Avatar pop-up screen in the MenuScreen state.
FR-INTERACT-PLACING	Interaction initiates a pop-up screen inside the GameScreen which freezes the character movement until exited through choices or by pressing E
FR-INTERACT-MESSAGE	When a player starts to interact with a building, there shall be a pop-up with text and choices
FR-MENU-SAVES	No class for saving the game. This was an intentional choice.
FR-MENU-PAUSE	While in GameScreen, Window escapeMenu allows the player to escape to MenuScreen by pressing Esc key followed by the exit button
FR-SLEEP	EventManager checks time of day before allowing activity. If 16 hours have passed all activities except sleeping are locked.
FR-ENERGY-RESTORE	Energy class and events can be seen in GameLogic.
FR-ENERGY-LIMIT	EventManager checks energy class for energy value
FR-WEEK	Day class, when on 7th day and time in Time class gets to 24 hours game will stop
FR-TIME	Activity class has amount of time it uses up which increases time in time class Dialogue allows
FR-USERNAME-LIMIT	In the state diagram, this FR ID can be seen during EnterName state.

References

[1] R. Wirfs-Brock. (2006, Jul.). A Brief Tour of Responsibility-Driven Design [Online]. Available: https://wirfs-brock.com/PDFs/A_Brief-Tour-of-RDD.pdf