UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

ENG1 Assessment 1
Group 18 - Octodecimal

Architecture

Group Members:

Izz Abd Aziz

Phil Suwanpimolkul

Tom Loomes

Owen Kilpatrick

Zachary Vickers

Michael Ballantyne

Introduction
For all of the diagrams created, we made use of Unified Modelling Language (UML) to allow us to easily construct and update architectural diagrams at the start, and throughout the software development process. We created these diagrams within the PlantUML Gizmo add-on to Google Docs, allowing us to embed our diagrams within our architecture documents.

CRC Cards
Before we began designing our diagrams, we created CRC cards to identify all potential key components of the game, and how they would interact with each other. Another reason we did this was to highlight the responsibilities that we initially thought each potential class should have. Using these cards gave us an initial idea of what our game's architecture may look like, and allowed us to create a simple framework that we knew would be altered largely during the development process. These cards can be seen on our website.

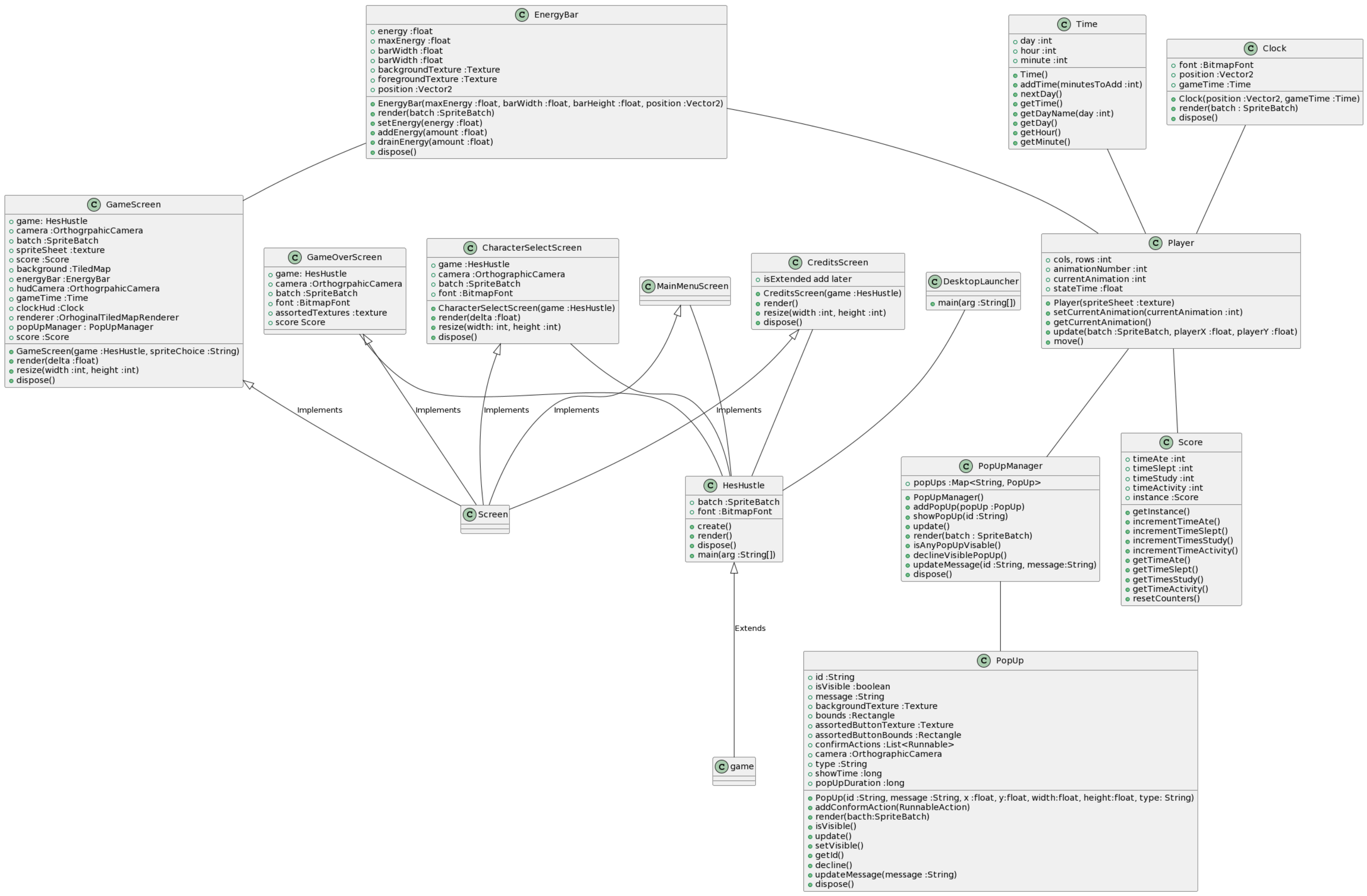
All CRC cards can be found here at:
https://eng1team18.github.io/website/architecture/CRC_Cards.pdf

Class Diagram
Initially, we created a rough class diagram based on: the requirements of the game, the CRC cards we had created and our pre-existing knowledge of how to implement them. Class diagram 1 shows each main component of the game as a class, each with properties we'd expect them to have and how we assumed each class would interact with each other. It includes abstract classes Screen (A prebuilt LibGDX class) and Place (an abstract class we would have to build ourselves and reuse) It served as a good reference for the development of the initial low fidelity prototype but there were things that had to change, as we discovered once we started development. For example, we ended up not adding a volume slider as our game didn't have sound. Additions also had to be made such as screens and renderers, things we didn't initially know the game needed and weren't shown in the first class diagram.

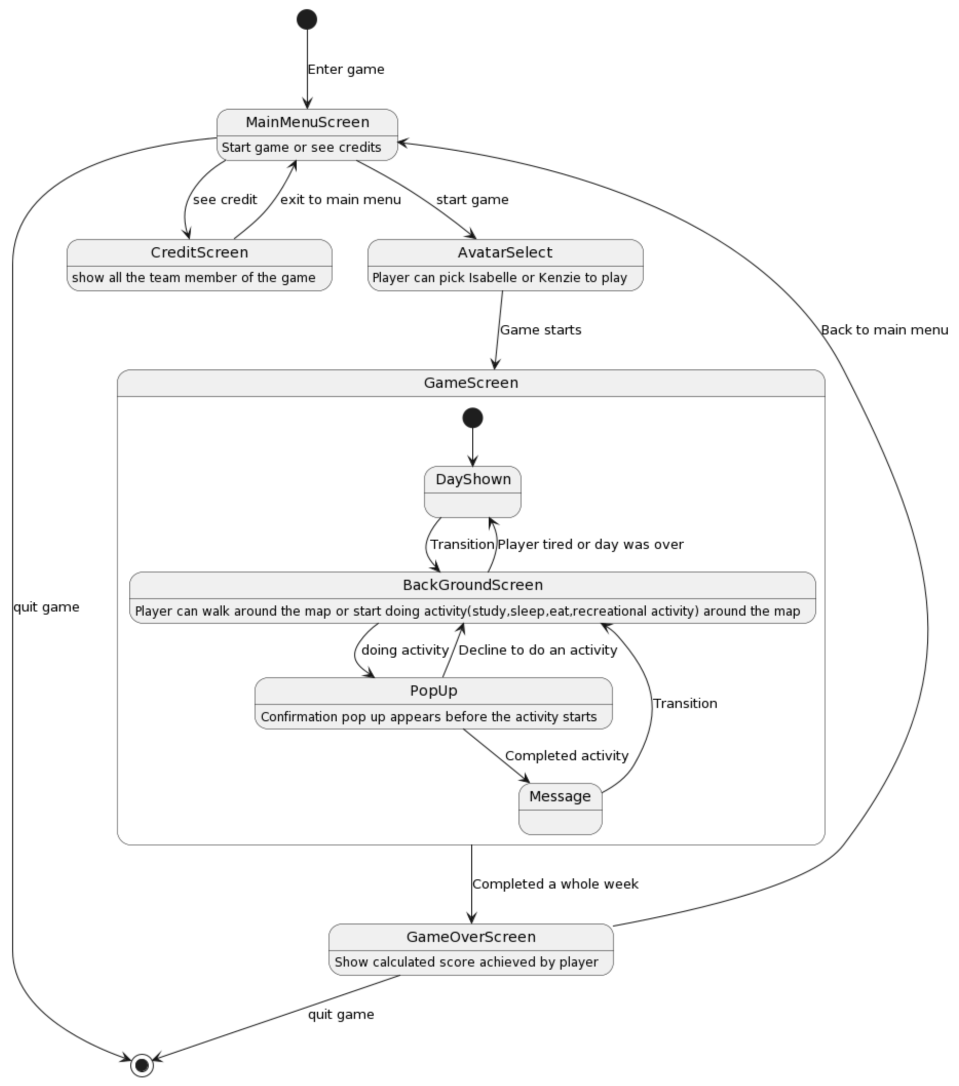
Moving on from the low fidelity prototype, we had our first client meeting where we were told new requirements and were given a more detailed vision of what the game should become. From this we created a new class diagram, each member of the team on implementation was given specific and mostly isolated tasks to work on which would lead to a class diagram without many associations at the moment (See ClassDiagram2).

Once we'd developed most of the features we managed to tie them together and did a little fixing of merge issues. What we were left with was ClassDiagram3

All diagrams and their interim iterations can be found at:
https://eng1team18.github.io/website/architecture.html


EnergyBar
energy :float
maxEnergy :float
barWidth :float
barWidth :float
backgroundTexture :Texture
foregroundTexture :Texture
position :Vector2
EnergyBar(maxEnergy :float, barWidth :float, barHeight :float, position :Vector2)
render(batch :SpriteBatch)
setEnergy(energy :float)
addEnergy(amount :float)
drainEnergy(amount :float)
dispose()
Time
day :int
hour :int
minute :int
Time()
addTime(minutesToAdd :int)
nextDay()
getTime()
getDayName(day :int)
getDay()
getHour()
getMinute()
Clock
font :BitmapFont
position :Vector2
gameTime :Time
Clock(position :Vector2, gameTime :Time)
render(batch : SpriteBatch)
dispose()
GameScreen
game: HesHustle
camera :OrthogrpahicCamera
batch :SpriteBatch
spriteSheet :texture
score :Score
background :TiledMap
energyBar :EnergyBar
hudCamera :OrthogrpahicCamera
gameTime :Time
clockHud :Clock
renderer :OrhoginalTiledMapRenderer
popUpManager : PopUpManager
score :Score
GameScreen(game :HesHustle, spriteChoice :String)
render(delta :float)
resize(width :int, height :int)
dispose()
GameOverScreen
game: HesHustle
camera :OrthogrpahicCamera
batch :SpriteBatch
font :BitmapFont
assortedTextures :texture
score Score
CharacterSelectScreen
game :HesHustle
camera :OrthographicCamera
batch :SpriteBatch
font :BitmapFont
CharacterSelectScreen(game :HesHustle)
render(delta :float)
resize(width: int, height :int)
dispose()
MainMenuScreen
CreditsScreen
isExtended add later
CreditsScreen(game :HesHustle)
render()
resize(width :int, height :int)
dispose()
DesktopLauncher
main(arg :String[])
Player
cols, rows :int
animationNumber :int
currentAnimation :int
stateTime :float
Player(spriteSheet :texture)
setCurrentAnimation(currentAnimation :int)
getCurrentAnimation()
update(batch :SpriteBatch, playerX :float, playerY :float)
move()
Implements
Implements
Implements
Implements
Implements
Screen
HesHustle
batch :SpriteBatch
font :BitmapFont
create()
render()
dispose()
main(arg :String[])
Extends
game
PopUpManager
popUps :Map<String, PopUp>
PopUpManager()
addPopUp(popUp :PopUp)
showPopUp(id :String)
update()
render(batch : SpriteBatch)
isAnyPopUpVisable()
declineVisiblePopUp()
updateMessage(id :String, message:String)
dispose()
Score
timeAte :int
timeSlept :int
timeStudy :int
timeActivity :int
instance :Score
getInstance()
incrementTimeAte()
incrementTimeSlept()
incrementTimesStudy()
incrementTimeActivity()
getTimeAte()
getTimeSlept()
getTimesStudy()
getTimeActivity()
resetCounters()
PopUp
id :String
isVisible :boolean
message :String
backgroundTexture :Texture
bounds :Rectangle
assortedButtonTexture :Texture
assortedButtonBounds :Rectangle
confirmActions :List<Runnable>
camera :OrthographicCamera
type :String
showTime :long
popUpDuration :long
PopUp(id :String, message :String, x :float, y:float, width:float, height:float, type: String)
addConformAction(RunnableAction)
render(bacth:SpriteBatch)
isVisible()
update()
setVisible()
getId()
decline()
updateMessage(message :String)
dispose()

State Diagram

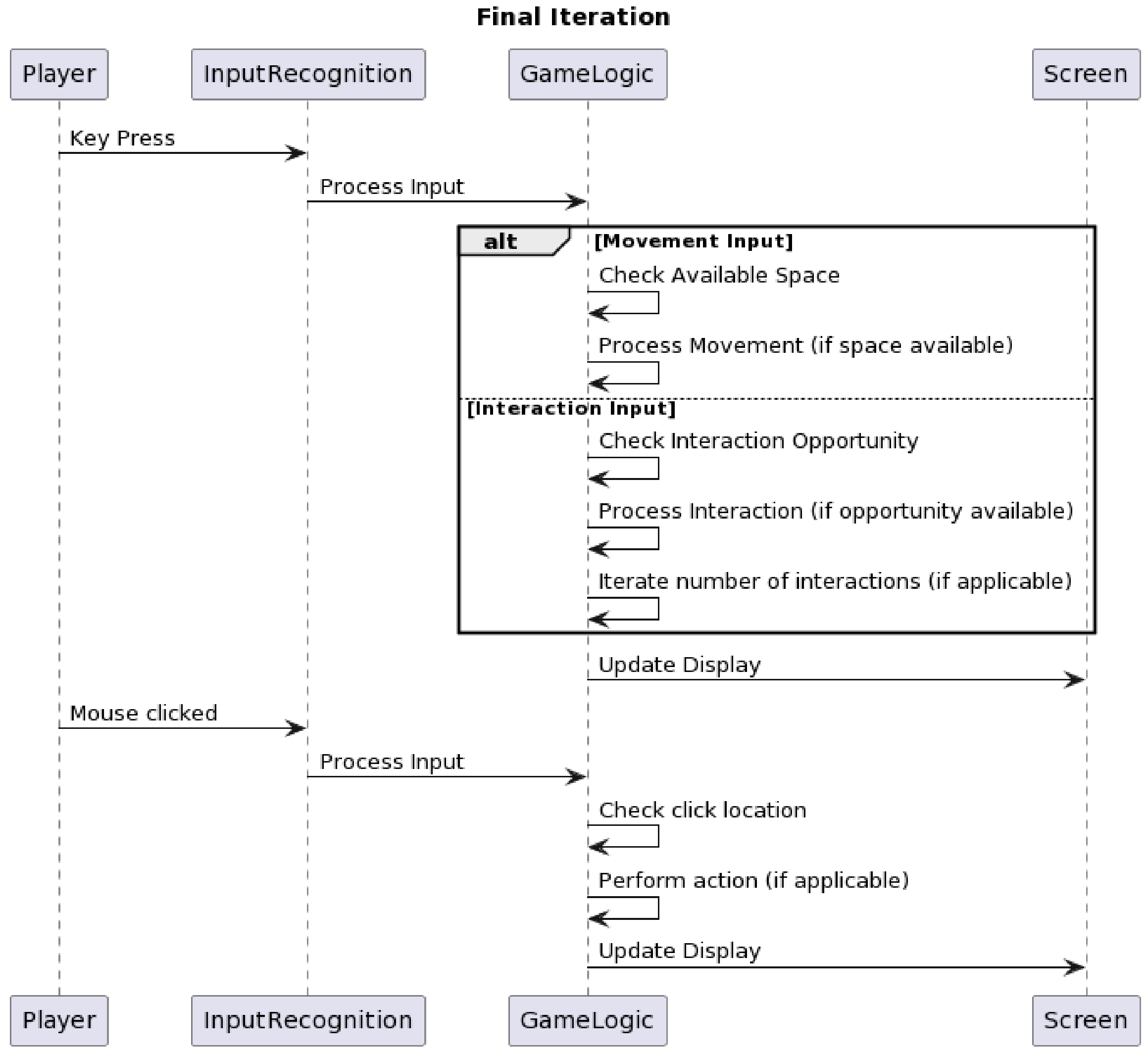

Enter game
MainMenuScreen
Start game or see credits
see credit
exit to main menu
start game
CreditScreen
show all the team member of the game
AvatarSelect
Player can pick Isabelle or Kenzie to play
Game starts
Back to main menu
GameScreen
DayShown
Transition
Player tired or day was over
BackGroundScreen
Player can walk around the map or start doing activity(study,sleep,eat,recreational activity) around the map
quit game
doing activity
Decline to do an activity
PopUp
Confirmation pop up appears before the activity starts
Transition
Completed activity
Message
Completed a whole week
GameOverScreen
Show calculated score achieved by player
quit game

When a player enters the game,the MainMenuScreen 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 and credits. The first iteration of our state diagram was a simple diagram that showed the loop between CreditScreen and MainMenuScreen. Initially, we are not planning to have AvatarSelect when you start a game but after our client interview it became required to have a selection of avatars to choose when you want to start the game. Then, the state has the second iteration which iterates from AvatarSelect until GameOverScreen which shows overall gameplay of Heslington Hustler. 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 or quit the game.

Sequence Diagram



We made use of a sequence diagram in order to visualise the relationship between the user and the game. This is done by working out the possible user inputs, and following that input through the game to see how it is processed by the system, from the key press all the way through to a character moving on the screen. The reason for this is to ensure that the game follows the correct procedure to accept the input, and then the logic within the game guarantees that correct result based off of the input, followed by the right display on screen.

The first iteration of our sequence diagram was a simple diagram that showed the relationship between the user pressing a key and the system checking if the key was valid before updating the screen. This would either move the character or allow the character to interact with something within the game, satisfying the specific user requirements that specified this (for reference requirements with ID FR_MAP_MOVEMENT and FR_TASKS_INTERACT for both movement and interaction respectively).

This diagram was then updated after our client interview, with our updated requirements helping to form the second iteration. With the addition of further stages of validation within the GameLogic phase, representing the addition of things like collisions, something brought up in the interview, the diagram is able to more accurately show the process within the GameLogic where it is checked if a user is able to move into a space after a key is pressed, and the character is then moved and the screen updated is there is a space to move into. Similarly, for interactions, the GameLogic checks if there is a relevant building or area nearby for a character to interact with if the interaction button is pressed.

The third iteration of the diagram was created during the implementation stage of the project. During implementation, it was decided that the menus of the game would be navigated using mouse clicks rather than key presses. This meant the diagram had to be updated to reflect this new way of the user interacting with the game. To do this, a new branch was added from Player to InputRecognition to represent a mouse click. This was then followed up with the input being processed, and the location of the click being checked to ascertain whether or not a button was pressed and which button was pressed, so that the appropriate action can be taken as a result of the button being clicked. Finally, the screen is then updated to reflect these changes.