CHANGE LOG

Team 20: Tequila Mockingbird

Exam Numbers: Y3872854, Y3877651, Y3872779, Y3873031, Y3869788

Approach to Change Management

For managing changes we first decided to clearly define what each of the requirements will be and then distributed them to sub-teams within the group. The sub teams were originally planned to be Oliver and Ben, Lucas and Ayyesha, Jay and Matthew and were planned as such that if any one member for whatever reason was not submitting work then there would be another team member to fulfill their sub-teams work.

The main goal with changes to implementation going forward was to make sure that integrity of the original code base was maintained and at all times minimal changes to the architecture of the program were made. Changes were planned ahead of time with consideration to how the program already functioned, making sure that similar approaches were used and that minimal impact on other sections of the code base was created by each change.

Furthermore in this section of the assessment the whole team had a focus on using continuous integration. We mainly implemented the philosophy of continuous integration manually by simply communicating to each other when there was a new commit to the GitHub repository and used frequent integration with good documentation and comments. We also implemented a system to manage continuous integration for us called CircleCI that handled the tests for us and provided a report of which tests passed and failed on each commit of the code to make sure that each commit was correct.

With regard to testing, each individual team was responsible for manually and visually testing each section of the code that they were changing by running through the game each time a change was made and reporting any bugs that could visually be found to the group or fixing them before committing the code. As automatic testing does not lend itself to a “game” type application most of the code was repeatedly tested using visual tests to see that each specific functionality of the game worked as intended.

The plan used going forward with completing the project was using an agile methodology meeting twice a week; once on Thursdays for scrums as a whole group and once on Saturdays in the sub-teams to work together and plan what work could be done by individual developers. The Thursday scrums were mainly used for analysing what requirements had already been completed, planning the next stages of the project and writing up sections of documentation for the deliverables.

Changes to Requirements

When we pricked up the project almost all the requirements in the original assessment brief had already been completed: only the requirement that the game displays a podium with winners and results was not completed.
The changes to the requirements that we have to fulfill for assessment 2 are as follows:

User Requirements

ID	**Description**	**Priority**
UR_DIFICULTY_CHOICE	The user should be able to select a difficulty before the game starts to change the number of obstacles and opponent difficulty.	High
UR_BOAT_ BONUS	There are bonuses that the boats can collide with to gain a boost to their stats and give an advantage. (Must be at least 5 different ones)	High
UR_SAVE_ GAME	The user can pause the game and choose to exit, saving the state of the game to play later.	High
UR_LOAD_ GAME	The user can select to play from a previously saved state.	High
UR_PODIUM	The user can view the the results at the end of the race on a podium	Very high

System Requirements

ID	**Description**	**Priority**
SY_DIFICULTY_CHOICE	Change the number of objects to be spawned and the scalar constant that the opponents stats are multiplied by.	High
SY_BOAT_ BONUS	Randomly spawn a collision object that when the boat hits gives an appropriate increase to speed, stamina, health, acceleration or turning.	High
SY_SAVE_ GAME	All variables and data structures relating to the state of the game should be able to be saved to a .json file on exiting the game.	High
SY_LOAD_ GAME	All variables and data structures relating to the state of the game should be loaded to a .json file if chosen to load from a save state.	High

SY_PODIUM	The results are loaded to a graphical display on a podium	Very high

Changes to Risk Assessment

The risk assessment was largely unaffected by the new requirements (difficulty choices, bonuses, saving/loading save game) however programming in a project that we did not previously work on or make entirely faces a large number of potential difficulties that exponentially increase the troubles with working on a programming project. As such it was important to update the risk assessment for the project we adopted. The following additions were made to the risk assessment:

ID	Type	Description	Likelihood	Severity	Mitigation
2.1	Project	Code base is incompatible with future requirements.	Medium	Very High	Plan ahead of ways to implement new requirements in ways that are possible with the code base.
2.2	Project	Code base uses unfamiliar libraries or coding methods.	High	Low	Research relevant libraries and work to properly understand the code base.
2.3	Project	Code base is incompatible with automatic testing.	Low	High	Write as many automatic tests as possible then test with manual tests for the rest of the requirements and testing of the code.
2.4	Project	Code base does not fulfill requirements expected to already be completed.	Medium	Medium	Fully understand the original and new assessment brief, plan and reassess a new list of requirements to fulfill before writing more code.

Changes to Architecture

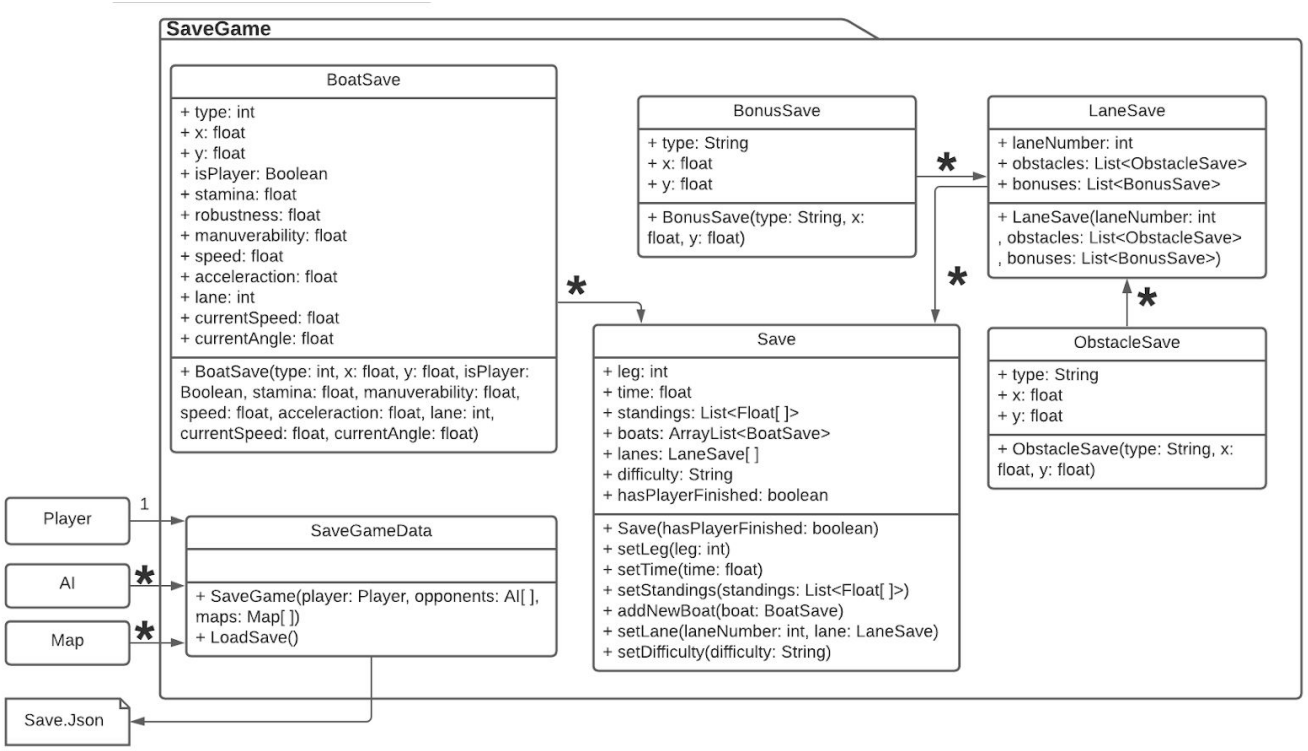
During the second assessment, the new requirements demanded that there were significant changes and additions to the architecture. There were oversights in some of the decisions when the first team initially created the game that, although didn't affect the original requirements, ment that adding to the game in future would demand significant changes.

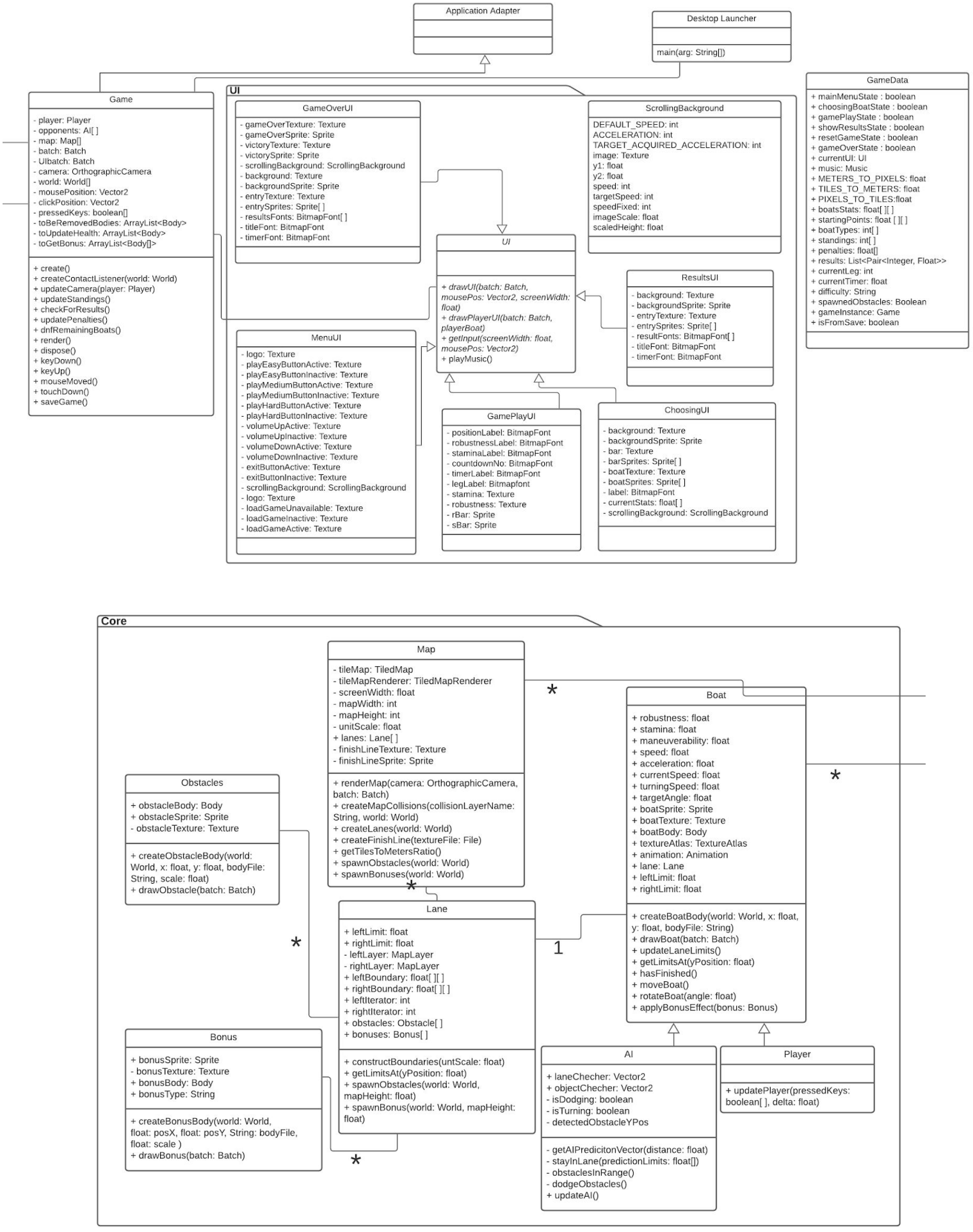
Some simple changes were required in cases such as changing the order the game is initialised to allow the user to select a difficulty level before the game generated the obstacles. More complicated changes were required in cases such as the save game functionality where every class relating to the state of the running game needed a corresponding “save” class to handle saving the file. See full details of what was changed in the architecture below.

Related Requirement	Change Made	Reason For Change	Impact on Project
UR_DIFICULTY_CHOICE	Removed the original play button, added three buttons: “play easy”, “play medium”, “play hard” to MenuUI.	Give the user the clearest and easiest method to change the difficulty.	≈None
SY_DIFICULTY_CHOICE	Changed Map.CreateLanes to change the number obstacles to be spawned depending on the difficulty.	Allow difficulty to affect the number of obstacles.	Medium, opponents hit too many obstacles.
SY_DIFICULTY_CHOICE	Made the opponents robustness stat scaling more dramatic the higher the difficulty.	The opponents hit too many obstacles on the higher difficulty causing DNF and automatic wins.	≈None
SY_DIFICULTY_CHOICE	Made the opponent's stats scale more depending on the difficulty chosen.	Allow difficulty choice to affect how difficult it is to beat the opponents.	≈None
SY_BOAT_BONUS	Added “Bonus” class.	Allowed for bonus functionality.	≈None
UR_BOAT_BONUS	Drawing pictures for bonuses inside “Game.render()”.	Allow users to see bonuses.	Added more junk to “Game.render”.
SY_BOAT_BONUS	Added list of bodies “toGetBonus” to “Game” class.	Allow boats to collide with bonuses and game to differentiate between bonus and obstacle.	Added more junk to “Game”class.
SY_BOAT_BONUS	Added method “Lane.SpawnBonus()” that is called in “Map.createLanes()”.	Allow control of obstacles to spawn in each lane.	≈None
SY_BOAT_BONUS	Added method “Boat.applyBonusEffect()”	Allow control of obstacles to spawn in each lane.	≈None

	that is called in “Map.createLanes()”.		
UR_SAVE_ GAME	Added PauseUI class to the UI package, opened using the “Esc” key.	Allow the user to stop the game, exit and save.	≈None
UR_SAVE_ GAME, UR_LOAD_ GAME	Added load button the MenuUI	Allow users to load the game from the main menu.	≈None
SY_SAVE_ GAME, SY_LOAD_ GAME	Added “Save.json” file that has all game data saved to and loaded from.	Creates files for saving and loading data to and from.	≈None
UR_PODIUM	Added results to “GameOverUI”.	Allows users to see final results on the final screen.	≈None

https://lucid.app/lucidchart/327aa72e-c5e6-4d48-a9a0-48f7d7997d4e/edit?beaconFlowId=AE1DA4B650D41B78&page=0_0#?folder_id=home&browser=icon

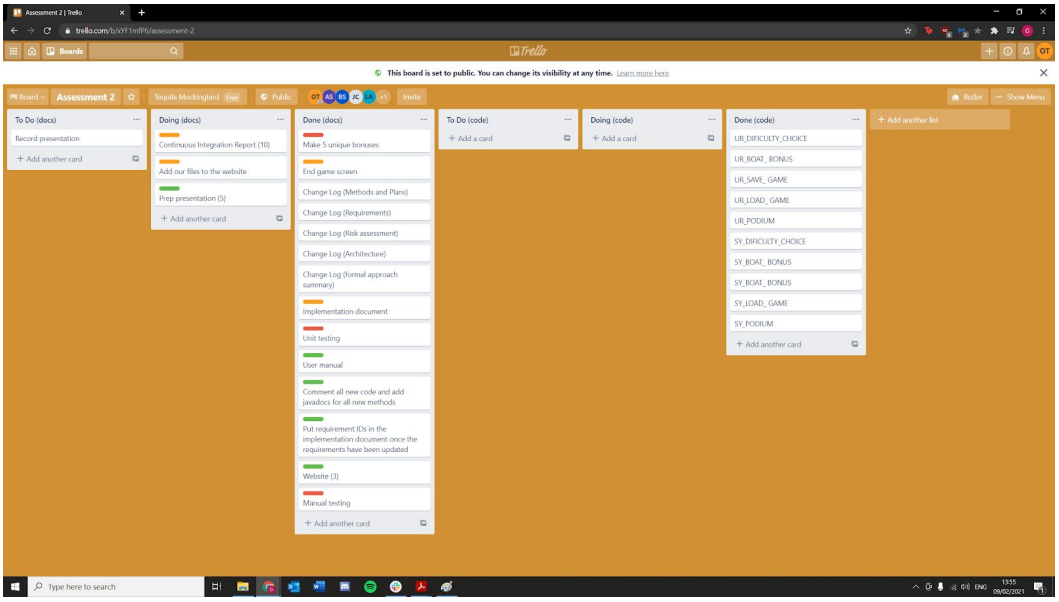



Application Adapter
Desktop Launcher
main(arg: String[])
Game
- player: Player
- opponents: AI[ ]
- map: Map[]
- batch: Batch
- UIbatch: Batch
- camera: OrthographicCamera
- world: World[]
- mousePosition: Vector2
- clickPosition: Vector2
- pressedKeys: boolean[]
- toBeRemovedBodies: ArrayList<Body>
- toUpdateHealth: ArrayList<Body>
- toGetBonus: ArrayList<Body[]>
+ create()
+ createContactListener(world: World)
+ updateCamera(player: Player)
+ updateStandings()
+ checkForResults()
+ updatePenalties()
+ dnfRemainingBoats()
+ render()
+ dispose()
+ keyDown()
+ keyUp()
+ mouseMoved()
+ touchDown()
+ saveGame()
UI
GameOverUI
- gameOverTexture: Texture
- gameOverSprite: Sprite
- victoryTexture: Texture
- victorySprite: Sprite
- scrollingBackground: ScrollingBackground
- background: Texture
- backgroundSprite: Sprite
- entryTexture: Texture
- entrySprites: Sprite[ ]
- resultsFonts: BitmapFont[ ]
- titleFont: BitmapFont
- timerFont: BitmapFont
ScrollingBackground
DEFAULT_SPEED: int
ACCELERATION: int
TARGET_ACQUIRED_ACCELERATION: int
image: Texture
y1: float
y2: float
speed: int
targetSpeed: int
speedFixed: int
imageScale: float
scaledHeight: float
GameData
+ mainMenuState : boolean
+ choosingBoatState : boolean
+ gamePlayState : boolean
+ showResultsState : boolean
+ resetGameState : boolean
+ gameOverState : boolean
+ currentUI: UI
+ music: Music
+ METERS_TO_PIXELS: float
+ TILES_TO_METERS: float
+ PIXELS_TO_TILES:float
+ boatsStats: float[ ][ ]
+ startingPoints: float [ ][ ]
+ boatTypes: int[ ]
+ standings: int[ ]
+ penalties: float[]
+ results: List<Pair<Integer, Float>>
+ currentLeg: int
+ currentTimer: float
+ difficulty: String
+ spawnedObstacles: Boolean
+ gameInstance: Game
+ isFromSave: boolean
UI
+ drawUI(batch: Batch, mousePos: Vector2, screenWidth: float)
+ drawPlayerUI(batch: Batch, playerBoat)
+ getInput(screenWidth: float, mousePos: Vector2)
+ playMusic()
ResultsUI
- background: Texture
- backgroundSprite: Sprite
- entryTexture: Texture
- entrySprites: Sprite[ ]
- resultFonts: BitmapFont[ ]
- titleFont: BitmapFont
- timerFont: BitmapFont
MenuUI
- logo: Texture
- playEasyButtonActive: Texture
- playEasyButtonInactive: Texture
- playMediumButtonActive: Texture
- playMediumButtonInactive: Texture
- playHardButtonActive: Texture
- playHardButtonInactive: Texture
- volumeUpActive: Texture
- volumeUpInactive: Texture
- volumeDownActive: Texture
- volumeDownInactive: Texture
- exitButtonActive: Texture
- exitButtonInactive: Texture
- scrollingBackground: ScrollingBackground
- logo: Texture
- loadGameUnavailable: Texture
- loadGameInactive: Texture
- loadGameActive: Texture
GamePlayUI
- positionLabel: BitmapFont
- robustnessLabel: BitmapFont
- staminaLabel: BitmapFont
- countdownNo: BitmapFont
- timerLabel: BitmapFont
- legLabel: Bitmapfont
- stamina: Texture
- robustness: Texture
- rBar: Sprite
- sBar: Sprite
ChoosingUI
- background: Texture
- backgroundSprite: Sprite
- bar: Texture
- barSprites: Sprite[ ]
- boatTexture: Texture
- boatSprites: Sprite[ ]
- label: BitmapFont
- currentStats: float[ ]
- scrollingBackground: ScrollingBackground
Core
Map
- tileMap: TiledMap
- tileMapRenderer: TiledMapRenderer
- screenWidth: float
- mapWidth: int
- mapHeight: int
- unitScale: float
+ lanes: Lane[ ]
- finishLineTexture: Texture
- finishLineSprite: Sprite
+ renderMap(camera: OrthographicCamera, batch: Batch)
+ createMapCollisions(collisionLayerName: String, world: World)
+ createLanes(world: World)
+ createFinishLine(textureFile: File)
+ getTilesToMetersRatio()
+ spawnObstacles(world: World)
+ spawnBonuses(world: World)
Boat
+ robustness: float
+ stamina: float
+ maneuverability: float
+ speed: float
+ acceleration: float
+ currentSpeed: float
+ turningSpeed: float
+ targetAngle: float
+ boatSprite: Sprite
+ boatTexture: Texture
+ boatBody: Body
+ textureAtlas: TextureAtlas
+ animation: Animation
+ lane: Lane
+ leftLimit: float
+ rightLimit: float
+ createBoatBody(world: World, x: float, y: float, bodyFile: String)
+ drawBoat(batch: Batch)
+ updateLaneLimits()
+ getLimitsAt(yPosition: float)
+ hasFinished()
+ moveBoat()
+ rotateBoat(angle: float)
+ applyBonusEffect(bonus: Bonus)
Obstacles
+ obstacleBody: Body
+ obstacleSprite: Sprite
- obstacleTexture: Texture
+ createObstacleBody(world: World, x: float, y: float, bodyFile: String, scale: float)
+ drawObstacle(batch: Batch)
Lane
+ leftLimit: float
+ rightLimit: float
- leftLayer: MapLayer
- rightLayer: MapLayer
+ leftBoundary: float[ ][ ]
+ rightBoundary: float[ ][ ]
+ leftIterator: int
+ rightIterator: int
+ obstacles: Obstacle[ ]
+ bonuses: Bonus[ ]
+ constructBoundaries(untScale: float)
+ getLimitsAt(yPosition: float)
+ spawnObstacles(world: World, mapHeight: float)
+ spawnBonus(world: World, mapHeight: float)
Bonus
+ bonusSprite: Sprite
- bonusTexture: Texture
+ bonusBody: Body
+ bonusType: String
+ createBonusBody(world: World, float: posX, float: posY, String: bodyFile, float: scale)
+ drawBonus(batch: Batch)
AI
+ laneChecher: Vector2
+ objectChecher: Vector2
- isDodging: boolean
- isTurning: boolean
- detectedObstacleYPos
- getAIPredicitonVector(distance: float)
- stayInLane(predictionLimits: float[])
- obstaclesInRange()
- dodgeObstacles()
+ updateAI()
Player
+ updatePlayer(pressedKeys: boolean[ ], delta: float)
*
1

Changes to methods and plans

We planned to use an agile methodology using weekly scrum meetings and sprints in teams of two (Oliver and Ben, Lucas and Ayyesha, Jay and Matthew). The key tools used were:

- Visual studio for software development. Extensions to visual studio such as liveshare were vital for group work as being unable to meet in person meant that working on the same piece of code in a pair at the same time was impossible however by using liveshare we were able to work on the same set of files simultaneously.
- GitHub was used as a file storage and management system. The version control and branch management supplied by GitHub allowed multiple teams to properly work in separate versions of the program then merge into one main branch that the final code was on: https://github.com/LucasAxnix/ENG1-P2
- For communication between team members we continued using slack, a professional communication service often used by workplaces that allowed for “pinging” group members which also sent them an email so that there was more chance you could reach someone if you needed a more urgent response.
- A Trello board was created to plan and manage the workload by splitting tasks among sub teams of the group and then moving between sections of the trello board saying “to do”, “doing” and “done” to properly communicate to the rest of the group what work was yet to be completed. https://trello.com/b/xYF1mfP6/assessment-2



- CircleCI was used for automated continuous integration, this was done in order to check that whenever the code was updated that the code would not break and would still pass all written tests. This was implemented towards the

end of the project due to problems with Junit and libgdx working together and so was done manually up until that point.
https://app.circleci.com/pipelines/github/LucasAxnix

Through regular meetings and sprints we were able to quickly complete the changes to the code. By meeting regularly with the group we then planned finishing testing and documentation individually: