Deliverable: Architecture

Group number and name: Group 10 - Uptown Func()

Group member names: Freddie Cooper, Brandon Stahl, Matthew Tsang, Ryuchie Tjhuanda, Joel Weston, Ben Williams

All of our diagrammatic representations of the game's architecture can be found on our website, they are named Architecture figure A, Architecture figure B, Architecture figure C, Architecture figure D. All our diagrams were created in LucidChart, which is a cloud-based, drag and drop diagramming application.

We created four different diagrams, two structural and two behavioural. The structural diagrams we created were a UML class diagram and a UML component diagram. Our behavioural diagrams are a UML sequence diagram and a UML state diagram.

We decided to make a UML class diagram (which can be found on our website named 'Architecture figure D', it may require zooming in but the font size is 10pt so it should be fine to view) to represent the classes our implementation uses and the relationships between them. A UML class diagram models a system by focusing on the structure of the code and how objects interact with each other. We found that this diagram is a good way to simplify the complex design of our implementation, as by already knowing the structure of our code we could spot any potential issues. As well as this, it allowed the programmers to conceptualise how the code was going to be laid out before it was written. This would mean that there would be less implementation time spent on the basics of the system, and more spent on ensuring as many of our requirements are met. We found, whilst programming, that the UML class diagram also aided our programmers to create code that all linked well together. We were concerned that, due to having multiple people coding related classes concurrently, there may be issues when integrating. However, this did not happen as often as it may have done without the UML class diagram, as it reminded us which classes needed to work together and encouraged us to communicate.
Initially, we designed the UML class diagram without including any attributes or methods (this can be seen on our website, labelled 'Original UML class diagram'), as we found it was too difficult to predict what we would need in that much detail without having implemented anything at all. This allowed us to strip the structure of the code down to its basics, we had a diagram purely consisting of the classes that would be needed and how they were related to each other. As we progressed through the implementation phase, we gradually added detail to the diagram when we had completed a different section. Some relationships between classes changed as we made tweaks to how different parts of the code worked, and this was updated too.

Our other structural diagram was a UML component diagram (which can be found named Architecture figure A on our website), this represents the different components that make up our game and how they are associated with each other. As can be seen in the diagram, it simplifies the complex relationship between the main abstract components of our game. By using this diagram, our programmers were encouraged to make the code more modularised. As it was clearly to see which sections each part of the code would fall into. By making the code modularised, this aided reusing parts of our code (through inheritance), but also aided teamworking as we could assign each member of the team a different section of the diagram. This means that there is less risk of incompatibility between related modules, as the same person would be likely to be creating them. It also allowed us to see how they could work at runtime, which was important for us to be able to understand what different interactions might be going on whilst the game is in play.

This diagram has stayed consistent throughout since it is an abstract diagram of how the components interact and did not get complicated by any details added during implementation.

For our first behavioural diagram, we created a UML sequence diagram (which can be found on our website named Architecture figure B). This diagram is intended to explain how different layers of the game pass information to result in an output for the user. This was a very important diagram in creating the main processes that our game would follow, as it broke down how the interaction occurred between user input, user interface and the game engine. We represented the following processes: moving the sprite (which involved collisions and reaching the end), interactions, pausing and running out of time. Our programmers could then follow this guide for how the layers would interact, and this helped them to produce a game that worked efficiently. This diagram aided our debugging, since we could track where information was supposed to be being processed and sent from or to, and work out at which layer an error was occurring.
This diagram remained consistent throughout the project, the example method names used were not used in our implementation as these represent an oversimplification of the process that actually occurs when information is passed between different layers of the game.

The second behavioural diagram we employed was the UML state diagram. This diagram was found to be very useful to model the dynamics of the game's states. It successfully simplified the way that the user interface would change in response to various user actions. When creating the user interface, we often referred to this diagram to ensure that we had covered all of the required transitions between states. It was also vital when we were testing the project, as we could go through this diagram and ensure that, under the correct circumstances, the game would move between states when required. As well as this, by having all of the main state transitions mapped out, we could be sure that we had the fundamentals of the game added correctly.

Requirement Achieved	Class/es which satisfy/ies requirement	Description of what has been done
UR1 - Instructions	StartScreen PauseMenu	At the start of the game it tells the user the aim of the game and any other valuable information the user might want. The PauseMenu includes a button which shows the different controls that the user has and what to press.
UR2 - Events	BasicCharacter Book food lake	We have 3 different types of events that the user can experience. A book which reveals a part of the map to the user, food which increases the stamina of the player and a lake at the end which slows the player down.
UR3 - Pausing	PauseMenu GameController	Allows the user to pause the game to read the controls, resume the game and quit the game when needed. The timer also pauses.
UR4 - View	GameCamera	We created an orthographic camera which renders the map as the player moves around exploring the maze. The player can press TAB to access the whole map they have explored.
SR1 - Genre	The Game's assets	We wanted to make sure that people of all ages could play the game so we went with a retro theme. Each of the assets are family friendly to hopefully appeal to everyone of any age.
SR2 - Accessibility	The Game's assets	Each asset is clearly distinct from each other in both looks and colour. This means that people who do require accessibility needs are able to understand the game.
SR3 - Perspective	GameCamera	The user has a top down view made from our orthographic camera which means that they are able to see all the routes around them. We tried to strike a balance between having too little to see and too much to see where people could easily complete the maze.
SR4 - Scalability		Libgdx allows you to easily change the window that the game opens on which we set to 1280x720 by default windowed however the user can resize to whatever they see fit
SR5 - Runability		We tested running the project on different OS systems and they were able to work as intended on them.
SR6 - Maze Generation	GameWorld MazeLoader	We hand drew out 4 different mazes as a text file of 1s and 0s. These are then rendered in and shown when in view of the user.

SR7 - Timer	GameTimer	There is a 5 minute timer which is shown at the top of the window. This will cause the player to lose when the timer is fully up and is paused if the user is in the pause menu.
UR5 - Items	Book Food GoalMarker	As stated in UR2 with our events these Book and Food are interactable items which are consumed when the user interacts with them. Goal Marker is technically an item at the end which allows the user to escape and win the game.
UR7 - Sprinting	BasicCharacter Lake	By pressing shift with the character they are able to increase their move speed. The lake in the final area also slows the player's character speed when walking and sprinting.
UR8 - Stamina	BasicCharacter	The character has a set stamina which is slowly consumed when the player sprints. The stamina bar naturally regenerates over time so the player can sprint again whenever they would like.
SR8 - Losing	GameTimer LoseScreen	As stated in SR7, there is a 5 minute timer and at the end of the 5 minutes the game shows the user the lose screen. Currently that is the only way of losing.
SR9 - Themes	GameWorld MazeLoader	Our 4 different sections are each textured differently creating 4 different themes. At the start the player starts in the Computer Science building and then moves over to the library. This is why we added the book which reveals the part of the map there. They then move to the 3rd area which is the accommodation block before making it to the lake.