Component Design and GUI Report
TEAM HEC

Table of Contents

1. Introduction

1.1. Report Structure

This report will describe why we designed the graphical user interface (which will henceforth be referred to as GUI) our game the way we did and in brief describe how we did it. We distinguish between GUI style - the look and feel of the GUI and GUI structure - the methods behind how the section works. We will use notation throughout for methods and classes for example: **getStartMapObj()**. If we do not include parameters that does not mean they don't have none however if we think the parameters are important to explain we shall include them.

1.1.0 In General…

The project in general has two "ends". The front, which deals with UI, and the back, which handles the game object and player data.

GameScreen is the main class in the UI everything comes from there. All the sections are handled with managers that create the **Sprites**, **Labels** and **SpriteButtons** that make up the UI.
Goal Menu and **Player Goals** handle goal actors that represent the **Goal** objects in the backend.
Map Manager handles the map objects and interacts deeply with the backend **WorldMap**.
Card Hand is the handler for the 7 cards your player is allowed the **CardActors** represent **Card** objects in the backend.
Game_Shop uses the **Shop** in the Player package to get the values it need and implement purchases and sells done in the shop UI.
GameScreenUI simply handles the **gamescreen** UI elements and only uses the backend for current player name, score and resources values.

1.1.1 GUI Style

With the game being a railway game already, we decided that the railway style would be carried throughout the graphic design of the game. In particular we were influence by ticket design and underground maps, with the map taking on an abstract underground map style. We also employed skeuomorphism (making a design like the original thing) when designing the goals as we designed the goals to look like train tickets. This helps to make the goals seem familiar. Gill Sans was the typeface we decided to use throughout, as it is visually very similar to Johnston, the font used in the London Underground's branding.

1.1.2 GUI Structure

The structure of the overall design has **LocomotionCommotion** class as its base. This class switches between the **StartMenu** and **GameScreen**. The table below explains the basic high level structure of the program.

Locomotion Commotion								
Start Menu	**GameScreen**							
start Scene	**Starting Sequence**	**GameScreenUI**	**Map Manager**	**Goal Menu & Player Goals**	**Shop**	**Card Hand**	**Pause Menu**	**Train Depot**
Section :	**3.1**	**3.2**	**3.3**	**3.4**	**3.5**	**3.6**	**3.7**	**3.8**

GameScreen uses the above 8 other classes: these classes create all the UI elements in their corresponding groups.

Starting Sequence: This class shows the get started windows that take the user through the first few steps, selecting a station each and then opens up the rest of the game and starts the player's turn.

GameScreenUI: This class is the group of game screen objects like the top bar, resources bar and the in-game options. The **GameScreenUI** also handles the resources, score and player turn labels.

Game_Map_Manager: Used to group all the routing button and labels, the Train and station information windows and handle entering routing mode.

Goal_Menu & Player_Goals: Goal menu is used for getting new goals. **PlayerGoals** are the goals that a player currently owns.

Shop: A Shop UI for buying and selling resources

Card_Hand : Card hand is used to hold the player's card actors. This also allows the user to use the cards.

Pause_Menu: A simple menu that allows user to resume, save and exit game and change preferences.

Train_Depot: The Train depot has not yet been implemented. But should be used for buying and upgrading trains.

2. Main Menu

2.1 Main Menu GUI

The main menu consists of four options, "New Game", for starting a new game; "Load Game", for opening previously played games; "Preferences", for changing the settings for the game; and "How to Play", a user guide on how to play the game.

Clicking on one of them takes you to that screen by panning the camera along the edge of the line drawn and stopping at the new page. This creates a smooth transition between screens and makes the menu more visually pleasing. Each of these screens have a button to take you back to the main screen, with "Exit Game" closing the game.

The main path for the users into the game is moving to the "New Game" screen. Here they are presented with a variety of user options: game mode, player names and number of turns. Options that aren't selected are semi transparent, with selected options being a solid colour so it's obvious what the user has selected. The text fields have Player1 and Player2 prewritten to prompt them to fill them in.

The "How to play" menu option takes the users to a screen where they are walked through how the game works.

2.2 GUI design

The **StartMenu** has everything on one plane. There is the central view (Main Menu - Title), moving up the camera gives you the New Game view, left gives how to play view, right gives load view and down gives you the preferences view. The animations are done by starting an action that moves the camera every render until a distance has been reached where the act finishes.

Example Structure of StartMenu UI Sprites and SpriteButtons:

Section	Name	Type	Action (onClicked())
Main	Lines	**Sprite**	none
	Title	**Sprite**	none
	NewGame	**SpriteButton**	Move Camera Up to NewGame
	LoadGame	**SpriteButton**	Move Camera Right to LoadGame
	Preferences	**SpriteButton**	Move Camera Down to Preferences
	HowtoPlay	**SpriteButton**	Move Camera left to HowtoPlay
	ExitGame	**SpriteButton**	Exits Game
...	...	...	...

*As you can see we have Sprites that don't have any action and we have **SpriteButtons** that have **onClicked**() actions. All the game elements follow this formula. Sprites and SpriteButtons are explained in the **Architecture** document.*

3. Game Screen

3.1 Starting Sequence

3.1.1 Selecting Stations

Upon starting a new game the first player is instructed to select their start station. When they mouse over a station, the station circle texture is toggled to a larger size, which helps to make it clear that they are interactive and which station they are currently selecting. When pressed, the users are presented with overlay information when mousing over each station to inform them of the cost of the station and what resources it generates. After it is selected, the station is highlighted on the map by the circle being stroked by the player's colour. The different stroke colour helps the players distinguish which stations they own, which are owned by the other player and which are completely unowned.

3.1.2 Selecting goals

They are then prompted to select a goal from the goal screen to add to their list of goals.

It is important the users are shown how to select goals as this is a key part of the game and as such they are pointed to the goal screen when starting a new game.

3.1.3 Design Choices

The starting sequence is used to guide the user through the initialisation of the game. We need each player to select a station and highlight that the core of the game is to select goals from the goal screen by giving them a pointer in the right direction.

3.1.4 UI Structure

For the GetStartedWindow we use two custom textures. The first is just a GET STARTED box and the second includes the pointer. The starting sequence follows the following list of actions:

1. Ask player 1 to select user
    a. 1st train is selected
2. Ask player 2 to select user
    a. 2nd train is selected
3. We call the start game methods in **GameScreen** (which creates a core game) and then **Game_StartSequence** (which handles the UI) .
4. Display the **GetStartedWindow** with the pointer to tell the current player to selected there first goal.

3.2 GameScreenUI

3.2.1 UI Design

Player's score is displayed on the top bar at the center. When players have completed all the actions they wish to do on their turn, the can end their turn by pressing the "End turn" button found in the bottom right corner, below where the person whose turn it is is displayed again. You can access the Train Depot, shop and map info by pressing the buttons left of the end turn button. The player's current fuel, money and cards are displayed at the bottom of the main game screen:

3.2.2 Design Choices and UI Structure

All the game screen options are at the extremities of the screen because users find that the norm, users know that the top bar is generally where you find settings or pause buttons and this formula for formatting the game gives us the most space for us to put our 'game interface' in our case this is our map.

GameScreenUI initialises all these elements and manages the resource labels keeping them updated using its method **refreshResources**(). All the UI is built up of **Sprites**, **SpriteButtons** and **Labels**. Icons were used to represent money and fuel type rather than text as they would have taken up more room. Many of the user interface buttons toggle the visibility of more information or other interactive screens. The reason for this is that the user interface would be very small and cluttered if everything were displayed at once making it difficult to understand. It would also reduce the area of the map, which is the main part of the interface and with smaller user interface elements they would be harder to click on them as well.

3.3 Map Manager

3.3.1 UI Design

Once goals have been selected, the user then must route a train to its destination.

- The user assigns a goal to a train by clicking on an available train
- A menu appears showing train information, with a "Plan Route" button allowing you to enter routing mode for that specific train.
- Routing Mode is then entered, changing the appearance of the screen to a dark mode, making it explicitly clear.
    - The end turn button is also hidden so it cannot be clicked on while in routing mode to prompt users to finish their route first
    - In routing mode players cannot click on other trains
- Users can then select adjacent stations which have been toggled to use a slightly bigger texture
    - This implies they are clickable.
- Clicking an adjacent station will attempt to add the connection to your existing route, if you can afford the fuel to get there
    - If not, a prompt window appears displaying how much more Fuel they need to add the connection.
    - If they do have sufficient fuel, the amount of fuel needed is deducted from the total at the bottom of the screen and the all the route labels are updated

- White circles are then added to highlight the route taken by the train through the map so when the user exits and re-enters routing mode, the preplanned route is clearly visible
    - A red circle then passes over every individual white one to indicate the route direction.
- If the user presses **undo**, the latest connection is removed from the UI and the labels updated
    - If the train has already started a connection, it is impossible to remove from the route past the trains position
- If the user presses **cancel**, the UI removes as many connections as possible up to the trains position
    - This is the same as a repeated undo but a lot quicker
- If the user presses **abort**, the UI removes all connections and moves the train back to the last MapObj it passed - losing the player the progress they made
    - This could be used for when a train collision occurs and the train has to move back to the previous station or junction
- **Confirm** button exits routing mode, allowing the player to click on their other trains again or click the end turn button

The routing mode window also contains information about the length of a route, how much of the route is left, and how much fuel the route will cost and which type of fuel it will be. This allows the user to make tactical decisions about the shortest route to take and how much resources they'll use with this route. It also gives them an indication of how long it'll take to reach the destination with the route remaining.

3.3.2 Map_Manager Structure

Game_Map_Manager handles the map itself, the train and station information windows and the routing UI. **Map_Manager** has methods like **enter**/**exitRoutingMode** and **move**/**hide**/**showinfobox**.

3.4 Goal Menu

3.4.1 UI Design

The goal screen provides them with a list of goals to be selected from, up to three of which can then be chosen. Free spaces in the selected goals are represented via transparent tickets to imply that new goals can fill those spaces.

As previously mentioned the goals take the form of tickets. This follows the theme of trains and displays information you would find on a real rail ticket such as start and end destinations, additionally we have reward value, train type, start date(a deadline turn to start the goal) and route (cities the train must go through). This goals interface can be brought up later on upon the completion of goals in order to select new ones. Currently active goals can be seen by pressing the ticket button rather than the goal screen button.

3.4.2 Goal Menu Structure

The Goal Menu uses 3 main classes: **GoalMenu**, **PlayerGoals** and Goal Actor. Goal Actor represents every ticket object in the game. **PlayerGoals** are the **GoalActor's** owned by the player, Goal Menu manages the screen and the

9 new goals (tickets). The text on the Goals (tickets) are all Labels. They are placed together with two different indices an index 1-9 for **GoalMenu** and 1-3 in **playergoals**; the labels and the goals are all created when the game is run and to change them all you need is there index.

3.5 Shop

3.5.1 UI Design

Each turn the player receives resources from their station(s) and from completing goals, as well as spending fuel with train movement.

There is also a shop where resources can be bought and sold (for 70% of the buying price), which can be accessed by pressing the dollar symbol next to the end turn button.

3.5.2 Shop structure

The Shop has 3 sections: Buy, Sell and TrainStore (unimplemented). The items that you can buy/sell are given a class each. This was done because each item them has a picture, quantity and cost labels, add and minus buttons and buy/sell button. When you click the buy or sell button it interacts with the shop class to add or subtract resources and gold from the current player. The same images are used as at the bottom of the menu for consistency and player familiarity. The number of resources is still visible at the bottom of the page which allows users to see how much they have bought and how much they have available to sell.

3.6 Card Hand

3.6.1 UI Design

The card hand can be accessed by clicking show cards at the bottom of the screen. This displays the cards the current player has, up to 7 cards are displayed at once. Clicking on an individual will move the card up so it can be seen more clearly and a use card button will appear above the card, clicking this button will consume the card and carry out the appropriate effect. Again the images used for resource cards use the same images as the resource symbols for consistency.

3.6.2 Game_Card_Hand Structure

The card UI is based in the **CardHand** and the **Card_Actor** class. Card_Hand creates the 7 cards (blank) and puts them in their corresponding slots and you can add and remove cards using its **addCard()** and **useCard()** methods. When you select a card it moves itself up and any others down, **usecardbtn** also appears and pressing it will implement the **useCard()** method. This allows you to preview the card before using it and allows more cards to be displayed in a smaller space. Cards are easily distinguishable due to their colours.

3.7 Pause Menu

3.7.1 UI Design

The in-game menu can be accessed by pressing the menu icon in the top right of the screen. This pauses the game and allows the user to return to the main menu as well as providing similar options to the main menu:

A dark backdrop is also added and layered on top of all the other buttons so they are no longer clickable within the pause menu. This also helps to make the pause menu stand out.

3.7.2 Pause_Menu Structure

The class initialises the Sprites and SpriteButtons. Resume Game goes back in game, Save Game will save the game instance, Game Setting will open a further menu where the user can change the setting and MainMenu will set the screen back to the **startScene**.

3.8 Train Depot

The Train depot has not yet been implemented. Initial plans for the train depot are to provide a screen where players can purchase more trains and upgrade their current trains with better speed and fuel efficiency. Currently the train depot would be accessed from the shop or directly from the train depot button above the shop button.

4.1 Asset generation

4.1.1 Map Assets

The layout and textures for the map were drawn in Sketch 3, a vector graphics drawing package, to ensure that assets could be exported at a variety of sizes without impacting upon the quality of the assets produced. During the prototyping phase a more abstract subway-map like style was chosen.

Circles represent stations, with the station label above circle to let the know which station it is. Junctions are represented with a square. Connections between different places are denoted by different coloured lines, which have an in game function when buying stations. By aligning stations to a grid rather than matching them perfectly to where they are geographically the lines could be constrained to 45º angle, and the line between any two adjacent places does not change angle. By doing this it made animation of trains between cities easier as they always had a straight path that could be worked out using just the start and end points of the movement using a single vector between them, rather than having to calculate paths along irregular or curved paths using bézier curves. This implementation can be found within the architecture document in which the train animation and position is calculated by scaling a vector between two adjacent MapObjs.

When exporting the map it was separated out, with the station labels and lines rendered and exported together as one layer. The station circles and junction squares were then exported to a single circle and square, with larger versions for hover states. Stations were also exported with a colour stroke, representing when the station is owned by a player (blue for Player 1, orange for Player 2).

One of the issues we encountered was that the assets had initially been drawn at a size that did not completely fill the screen. This was easily fixable because they were produced as vectors, so the whole map could be exported at 1.3x the resolution. The hover states could also be exported at a larger size to emphasise them more.

Images were exported in the .png format to conserve their transparency.

4.1.2 Other Assets

All other asset were created in Adobe Illustrator which again uses vector graphics and are exported as individual assets in the png format.

4.2 Usage

The background of the map contains just the lines and station labels. The station dots all use the same texture and can be placed onto the map. The reason for keeping them separate is that it allows for the stations to alter their appearance on hover or purchase, whilst only needing a small set of images, rather than a unique one for each station or leaving the appearance unchanged. This also provides a click area for the user interaction. Trains are similarly rendered separately above to allow for movement and animation between stations.

4.3 In Game Screens

The in-game screens are the **GoalMenu**, **PauseMenu**, **Shop** and **TrainDepot**. These are groups of actors that we open and close. When we open **PauseMenu** for example we run through a loop of the actors in that section and make them all visible and because they are instantiated after main game screen they appear on top of everything. For us to be able to do this we fetch a stage start index and a stage end index, which are set when we create the group. This method of doing this was established very early in the program development and there are better ways, less messy ways to do this. We recommend using the “Scene” methods used in **StartMenu**. Scenes are explained in the ***Architecture Document***, this way of doing this is cleaner however we did not have time to change this ourselves as it was low priority.

5.0 User Requirements not met

User.UI.10 was the only user interface requirement not satisfied which is due to Events not being implemented. A hazard refers to anything that would cause a player to have to suffer a penalty or re-route a train. At the present time a player will never face a hazard hence why it was not added to the user interface.