CSCI205 Final Project

Team BabaIsUs

DESIGN MANUAL

I. Introduction

Baba Is Us is a parody of the awarded single-player puzzle video game Baba Is You. The game consists of three levels, each level contains some unmovable blocks and movable blocks that control the state of the game. There are two main different kinds of blocks; Entities and Words. Entities do not contain any properties on their own, this is where the Words come in. When a NounWord, ConnectorWord, and RuleWord all connect in this order respectively, the rule stored in the RuleWord is applied to all Entities corresponding to the NounWord used. The user's goal is to manipulate the game board until they can create a win condition for themselves.

For our version of the game, there are three main parts in our design: From the front end perspective, there is the game menu (what the user sees when they first start the game) and the game board (the actual game). We have two approaches to the game board, which we will do deeper in the object-oriented design section. From the back end perspective, there is the game manager that allows the transition between the GUIs. All GUIs are created using JavaFX. Aside from the main components, there are also the smaller GUIs such as the in-game GUI or the instruction GUI. We also included a music player to entertain the player.

II. User Stories

User stories		Status
As a user, I want to ...	Start the game with an options-menu	Complete
	Game rules explanation in options-menu	Complete
	Exit application in options-menu	Complete
	Have sound when the app starts	Complete
	Being able to edit the sound and visuals (brightness) when the app starts in option-menu	**Incomplete**
	Have a cool theme for the options-menu	**Incomplete**
	Being able to choose to play (go into play-menu)	Complete
	In play-menu, able to choose different levels (at least 3)	Complete
	In play-menu, able to exit the play-menu into options-menu	Complete
	In play-menu, also have a cool theme	Complete
	In play-menu, I can move around with my character (or arrow) to	**Incomplete**

	choose the level	
	In level, I can visually see the game board	Complete
	In level, there are objects on the board	Complete
	In level, have words on the board	Complete
	In level, have adjacent words placed together generate game rules if they are valid sentences	Complete
	In level, I can move pieces around the board	Complete
	In level, have win conditions for each board	Complete
	In level, have lose conditions for each board	Complete
	In level, I can quit in the middle and go to level-menu or quit application altogether	Complete
	In level, I can edit the board visually (brightness/ ...) to accommodate vision, or edit the game music	**Incomplete**
	In level, I want to be able to customize the theme	**Incomplete**
	In level, I can get a hint when stuck	Complete
	In level, scan the game board to change rules every time pieces are moved	Complete
	After level, Have confetti to celebrate + yay when I win/ boo or sad sound when I lose	**Incomplete**
	After level, regardless win/lose , I can choose to play again, or go to level-menu/ options-menu	Complete

Figure 1. The user stories of Baba Is Us

Our user stories are collected from our team discussion where each member acted as the customer and asked what they wanted from the game. Our initial goal was to collect as many user stories as possible so that we could have more information to work with, but we realized that having too much information created confusion between what we wanted to do and what we could do. From then, our priority was to create a functional game first, before adding complex and time-consuming features.

As the programming part progressed, we realized that some of the user stories were very hard to implement, and most of our incomplete user stories are related to customizing our game. Adjusting the brightness of the GUI was a challenge, as the code was complex, and arguably was not necessary as the user can already adjust the brightness of their own screen. Another user story we skipped was the level selection part, as it created many conflicts, and we opted for a more simpler approach of moving to the next level every time the player completes the current level. Overall, our product satisfies almost every user story.

III. Object oriented Design

Our initial design idea was to have two main GUIs to display the complete game which are the main menu which displays the rules and customization options and the game board that displays the character and objects with some method to control their interactions. There will be a controller to control the transition between the scenes. We also have a general object model where every object can apply it, and a factory to generate the objects. To demonstrate our idea, we have included our initial CRC cards below:

Main Menu GUI	
Responsibilities	**Collaborations**
Go to game	Game GUI
Display rules	
Settings	
Exit	

Game GUI	
Responsibilities	**Collaborations**
Display objects	General Objects Baba Factory
Display movable character	General Objects
Adjust game settings	Ingame menu GUI
Continuously update the state of the objects	
Keep track of the interactions between objects	General Objects

In-game Menu GUI	
Responsibilities	**Collaboration**
Go back to game	Game GUI
Go back to main menu	Main menu GUI
Adjust block colors	General Objects
Get hints for the levels	
Restart the game	Game GUI

Scene Controller	
Responsibilities	**Collaborations**
Control transitions between GUIs	Game GUI Main Menu In-game GUI

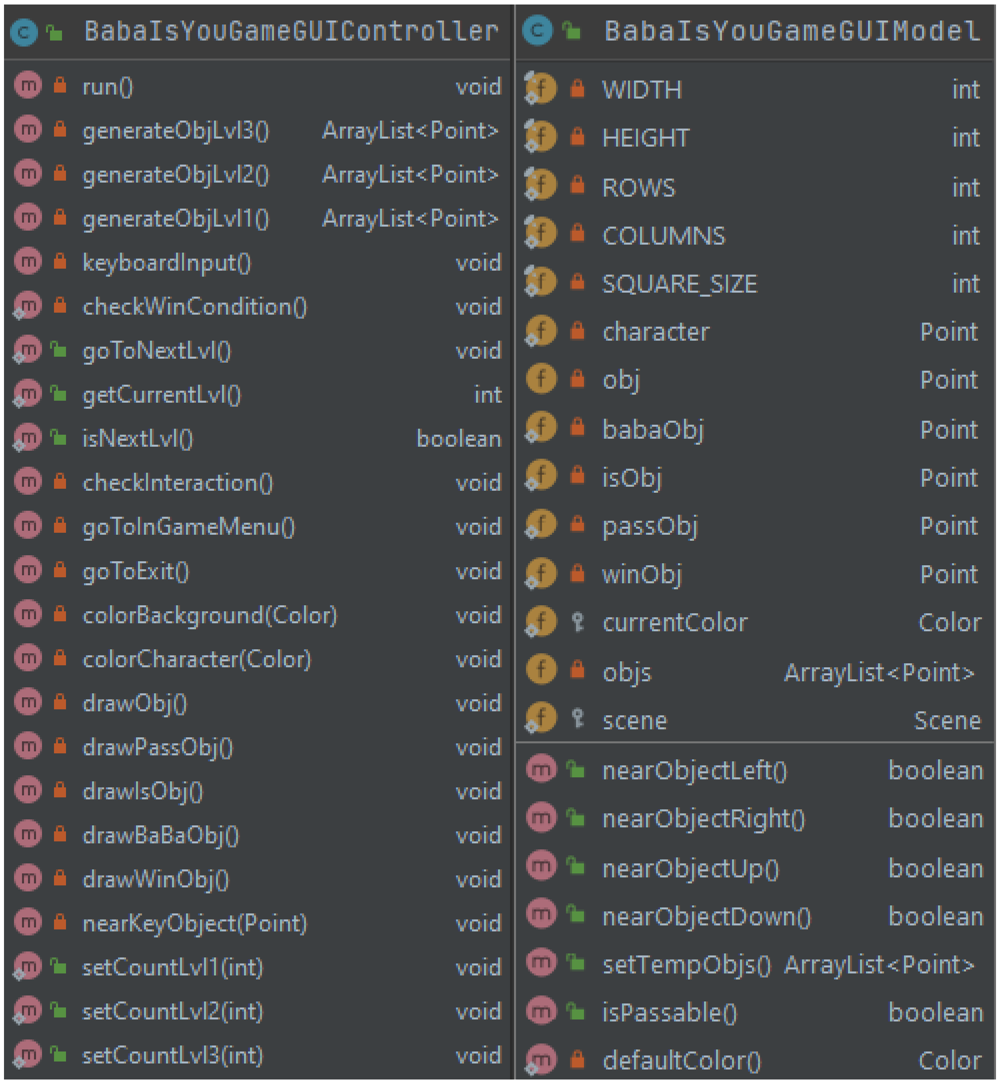
General Objects	
Responsibilities	**Collaborations**
Have a location	
Move around the board	
Stores an image	
Get color	

Baba Factory	
Responsibilities	**Collaboration**
Generate objects	General Objects

Figure 2. The CRC cards of Baba Is Us

Since this was only the initial planning stage, we divided our work to increase our efficiency. In sprint 1 and 2, Anh Tran and Nguyen Nguyen were assigned to work on the GUIs, while Yuhan and Christiaan were assigned to work on the objects and factory. For the GUIs, we decided to divide every GUIs into separate packages and apply the MVC design pattern for easier management and scene transition between every GUIs. However, the design pattern changed our implementation of the objects a lot, as we realized that, in the MVC of the game GUI, we could have the objects displayed on the View, the logics and interactions controlled by the Controller, and the objects be contained by the Model. This led to our decision of having two approaches: Nguyen would be in charge of implementing a simpler version of the objects in the Model of the game GUI, while Yuhan and Christiaan would still work on the General Objects and the Baba Factory as it was more flexible and can potentially be more aesthetically pleasing as we could incorporate some images as well.

Approach 1:

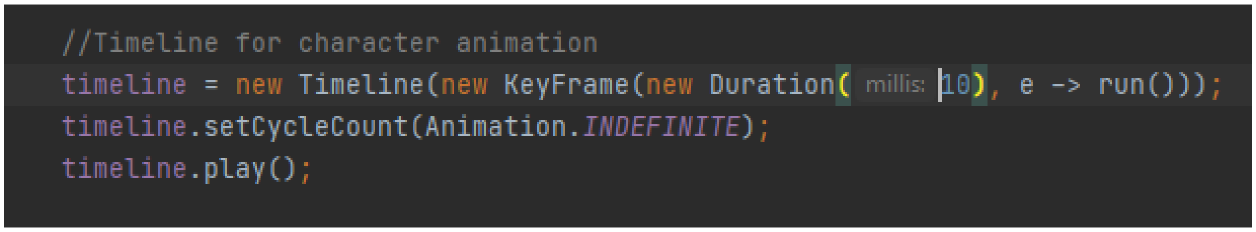


BabaIsYouGameGUIController	
run()	void
generateObjLvl3()	ArrayList<Point>
generateObjLvl2()	ArrayList<Point>
generateObjLvl1()	ArrayList<Point>
keyboardInput()	void
checkWinCondition()	void
goToNextLvl()	void
getCurrentLvl()	int
isNextLvl()	boolean
checkInteraction()	void
goToInGameMenu()	void
goToExit()	void
colorBackground(Color)	void
colorCharacter(Color)	void
drawObj()	void
drawPassObj()	void
drawIsObj()	void
drawBaBaObj()	void
drawWinObj()	void
nearKeyObject(Point)	void
setCountLvl1(int)	void
setCountLvl2(int)	void
setCountLvl3(int)	void

BabaIsYouGameGUIModel	
WIDTH	int
HEIGHT	int
ROWS	int
COLUMNS	int
SQUARE_SIZE	int
character	Point
obj	Point
babaObj	Point
isObj	Point
passObj	Point
winObj	Point
currentColor	Color
objs	ArrayList<Point>
scene	Scene
nearObjectLeft()	boolean
nearObjectRight()	boolean
nearObjectUp()	boolean
nearObjectDown()	boolean
setTempObjs()	ArrayList<Point>
isPassable()	boolean
defaultColor()	Color

Image 1. The class UML of the Controller (left) and the Model (right)

The code for the first approach is contained in the gameboard package. Using the MVC design pattern, every object and their interactions with each other are put in the model, including the character, the movable objects (Baba, Is, Win, Pass), and the unmovable objects (the orange blocks). The controller contains every method needed for these objects to interact with the keyboard input by the user, including moving the character, opening the in-game menu. It is also in charge of checking the current level, moving to a new level every time the user connects the Baba-Is-Win (or Purple-Blue-Pink) blocks together.

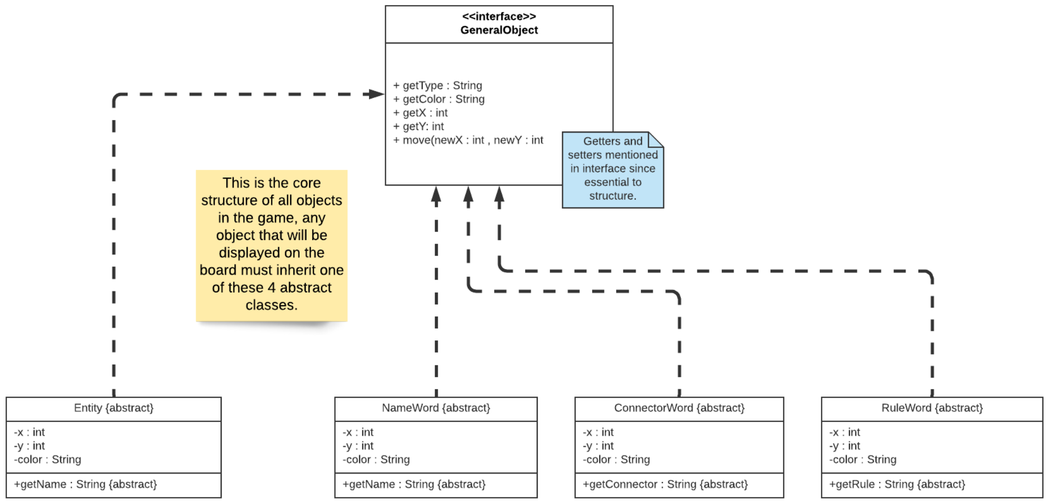


```
//Timeline for character animation
timeline = new Timeline(new KeyFrame(new Duration( millis: 10), e -> run()));
timeline.setCycleCount(Animation.INDEFINITE);
timeline.play();
```

Image 2. The timeline updating the game state every 10ms

The two most important pieces of code in the controller are the method **run()** and the variable **timeline**. **run()** calls every aforementioned method in the controller to ensure that everything runs, while **timeline** calls **run()** every 10ms, ensuring that the state of the game is updated. The view then displays the code, while the main calls it. This allowed us to create a simple version of the game that is fast, easy to implement, and presentation. As this was a simple version, we completed the implementation in sprint 3. We call this the back-up version, or the beta version of the game.

Approach 2:



<<interface>>
GeneralObject
+ getType : String
+ getColor : String
+ getX : int
+ getY: int
+ move(newX : int , newY : int
Getters and setters mentioned in interface since essential to structure.
This is the core structure of all objects in the game, any object that will be displayed on the board must inherit one of these 4 abstract classes.
Entity {abstract}
-x : int
-y : int
-color : String
+getName : String {abstract}
NameWord {abstract}
-x : int
-y : int
-color : String
+getName : String {abstract}
ConnectorWord {abstract}
-x : int
-y : int
-color : String
+getConnector : String {abstract}
RuleWord {abstract}
-x : int
-y : int
-color : String
+getRule : String {abstract}

Figure 3. The initial UML diagram for the core GeneralObject structure

The code of the second approach is stored inside the Rule package. The main idea of this approach is to utilize the General Object interface created by Christiaan and the Factory created by Yuhan. Even though the GeneralObject class hierarchy for this approach is quite simple at its core, it took a long time to figure out the best way to optimize the structure to work within a Factory and be able to apply essential game functionality easily. Originally this was implemented as a single abstract GeneralObject class that every object in the game inherited but this was very confusing to think about and needed to be changed a good amount. The structure we decided on was an overarching GeneralObject interface implemented by 4 abstract classes to split up into the 4 main object types, which then provides a simple and integrated template to code new objects easily. In sprint 3, after further implementation, we decided to drop the Factory class as it served no purpose, and decided to generate the objects when needed.

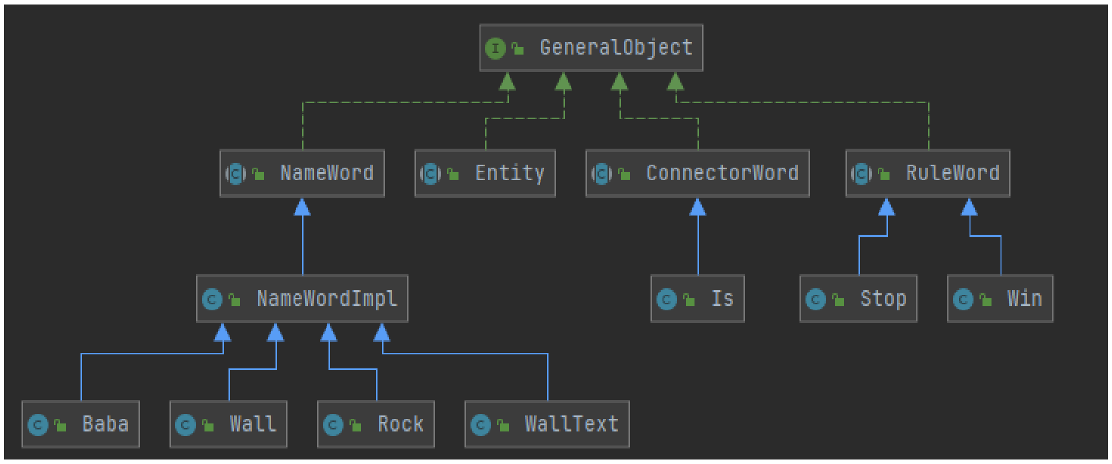


Image 3. The UML diagram for the core GeneralObject structure after implementation

As the first approach already served as the back-up plan, we decided to be more ambitious with the second approach, expanding the number of objects as it is more flexible and has the potential to be the main version if enough time is spent. From sprint 4 and onwards, Yuhan was the main person in charge of working on the objects and levels for the second approach with the help of other members. After careful consideration, we decided to have 4 NameWords (Baba, Wall, Rock, WallText), 1 ConnectorWord, and 2 RuleWord. The objects are then generated within the GUIs for each level, and we had a separate rule for each level as well to ensure maximum flexibility for the levels.

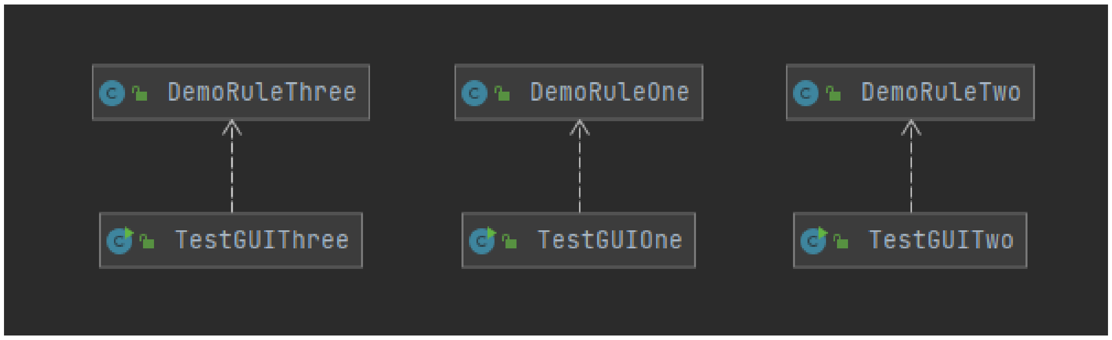
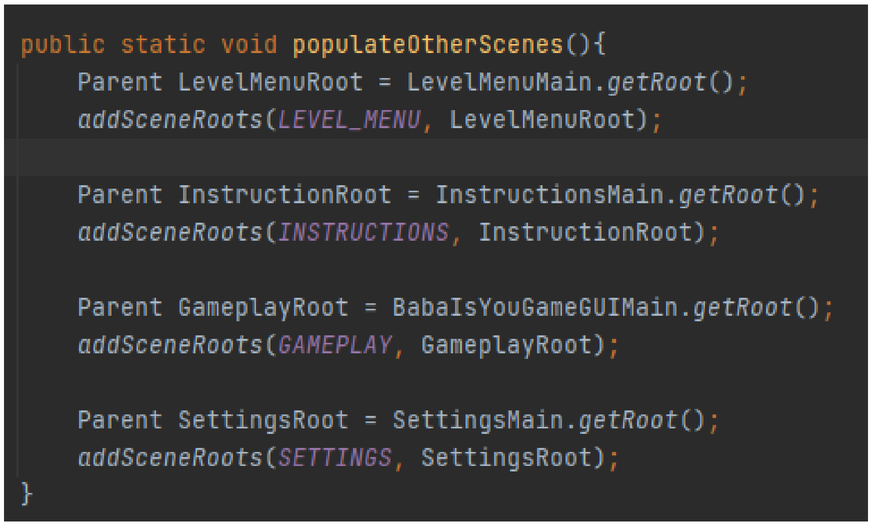


Image 4. The level structure of approach 2

Having these approaches in mind, we also needed a game menu where the players could read the instructions, go to the game, or exit the game. Additionally, realizing that we had many GUIs, another task was to connect the GUIs together. Duc Anh was in charge of creating the Menu GUI, music and other customizations, as well as connecting the GUIs together. As every GUIs is put in separate packages, he is also responsible for making sure that every time the user chooses an option that needs a transition to a new GUI, the transition would go properly. As the process of connecting the pieces together requires everything else to be done, this was done at the end of Sprint 4. The GUIs are connected by the gameManager package, and the transition is done by changing the current root to the root of the new GUI.



```
public static void populateOtherScenes(){
    Parent LevelMenuRoot = LevelMenuMain.getRoot();
    addSceneRoots(LEVEL_MENU, LevelMenuRoot);

    Parent InstructionRoot = InstructionsMain.getRoot();
    addSceneRoots(INSTRUCTIONS, InstructionRoot);

    Parent GameplayRoot = BabaIsYouGameGUIMain.getRoot();
    addSceneRoots(GAMEPLAY, GameplayRoot);

    Parent SettingsRoot = SettingsMain.getRoot();
    addSceneRoots(SETTINGS, SettingsRoot);
}
```

Image 5. The SceneManager of the gameManager package

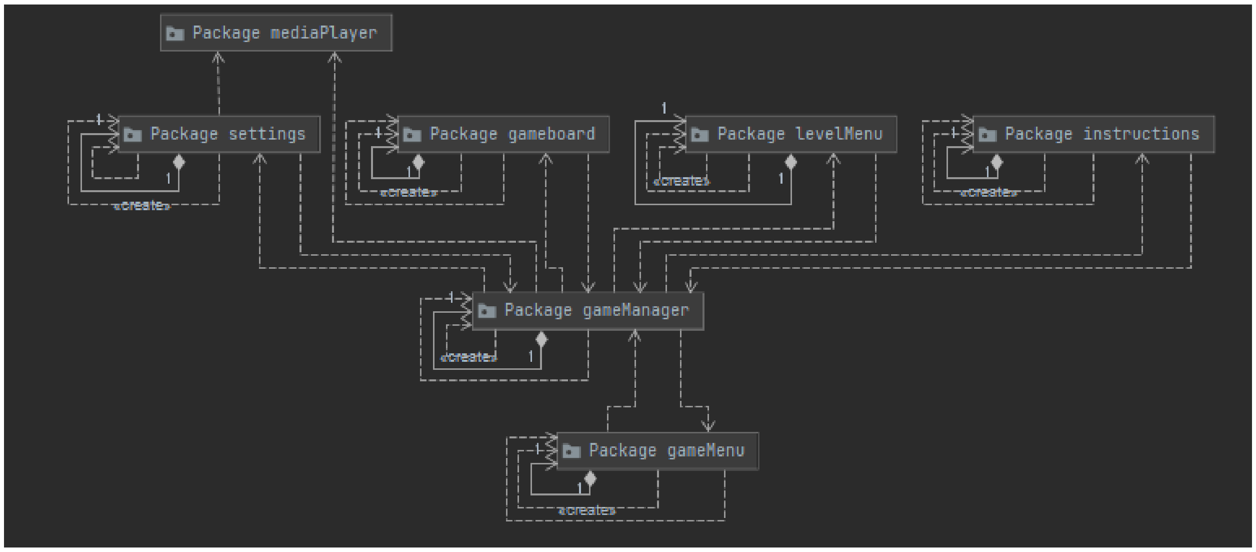


Image 5. The overall game structure from the packages point of view