CS319 Object-Oriented Software Engineering Final Report Iteration-1

RUSH HOUR

GROUP 3.B

Kuluhan Binici
Yusuf Dalva
Musab Gelişgen
Melih Ünsal
Ahmet Avcı

1. Introduction

Currently, there are huge chunks of programs missing from the project. One of these parts is Multiplayer game mode. This is simply because implementing the singleplayer game mode would get us started briefly with the project such that multiplayer game mode would be added later easily. They not only have similar characteristics (except for the shifting the game board part and having wildcards in the game) but also have the same dynamics, i.e., players just move some set of cars.

As spoken earlier, we are currently able to play the singleplayer game by choosing the singleplayer mode and selecting a map. Cars are placed according to predefined structure and the player can shift the cars accordingly.

Image for the cars also can be obtained from png files and there are many of them. However, choosing skins for the escape car has not been implemented yet. Some minor adjustments to the car skins will be made by using more beautiful png files and selecting a theme for the map will be added (which also will be made possible through png files).

2. Design changes

As high-level changes, in the User Interface part, we have done which we already have aimed to do and the single player version is also roughly done. However, the multiplayer mode, select theme, and select car pages were not created. On the other hand, the buttons for all the pages were created and their colors are changing intermittently. Right now, we focused on the visual details and the flow of the pages and the game logic. For now, we have completed Car objects. We can drag the cars so that they neither overflow of the map nor overlap with each other. There is also a music in the main menu which stops when the player selects the mode of the game. When the player completes the level, a clap is heard which indicated completion of the map.

In terms of low-level changes, class diagrams have already changed a lot even if the whole set of classes is not ready. We are now aware of the incomplete/missing parts from the Design Report-1 and now we are trying at our best to make up for these misjudgments. The class diagram of our program as of 18.11.18 (Many unimplemented classes are missing from this diagram, hence it is extremely incomplete at this point) can be seen in the **appendices** section. The most important reason of these low-level design changes is that we were not familiar with the game engine we are using, so we did not know it would be this different from the plain Java AWT/Swing libraries without extensive search and learning process. Now that we are versed, we are ready to go.

3. Lessons Learnt

One of the main problems in the project was of course lack of communication and knowledge. Although we scheduled regular meetings with the team members, some content kept getting away from our attention and we forgot to mention some of our ideas with others. This resulted in two very different class diagrams from each other (the class diagram from the design report and the current project's class diagram). However, we will not let this divergence get ahead of us and complete the project with optimal conditions.

The second lesson we learned might be not being intimidated by the number of requirements we put on the analysis report because we realized that it is easy to complete many of the requirements in a couple of hours. Although the features are incomplete we truly believe in ourselves that we can overcome these difficulties with teamwork.

4. User's Guide

4.1 System requirements & installation

Recommended System Requirements:

- A Desktop Computer or Laptop (Windows 8 or higher / Linux / MacOS X.
- **128 MB** memory or higher.
- The computer screen size should support at least **640 x 480**
- An Intel processor with **1.1 GHz** or higher

Installation:

The setup steps expect git to be installed on the system.

First, check out the repository. This process might expect you to have access to this private project, so make sure you are logged in to your GitHub account from your terminal.

```bash
git clone https://github.com/musabgelisgen/RushHour
```

Then open this project with an IDE (preferably Eclipse since this is also our development environment).

Compile and run the project using the IDE and enjoy.

p.s. These installation steps are just used for demonstration purposes. We plan on creating a .jar file to open game easily when the project is finished.

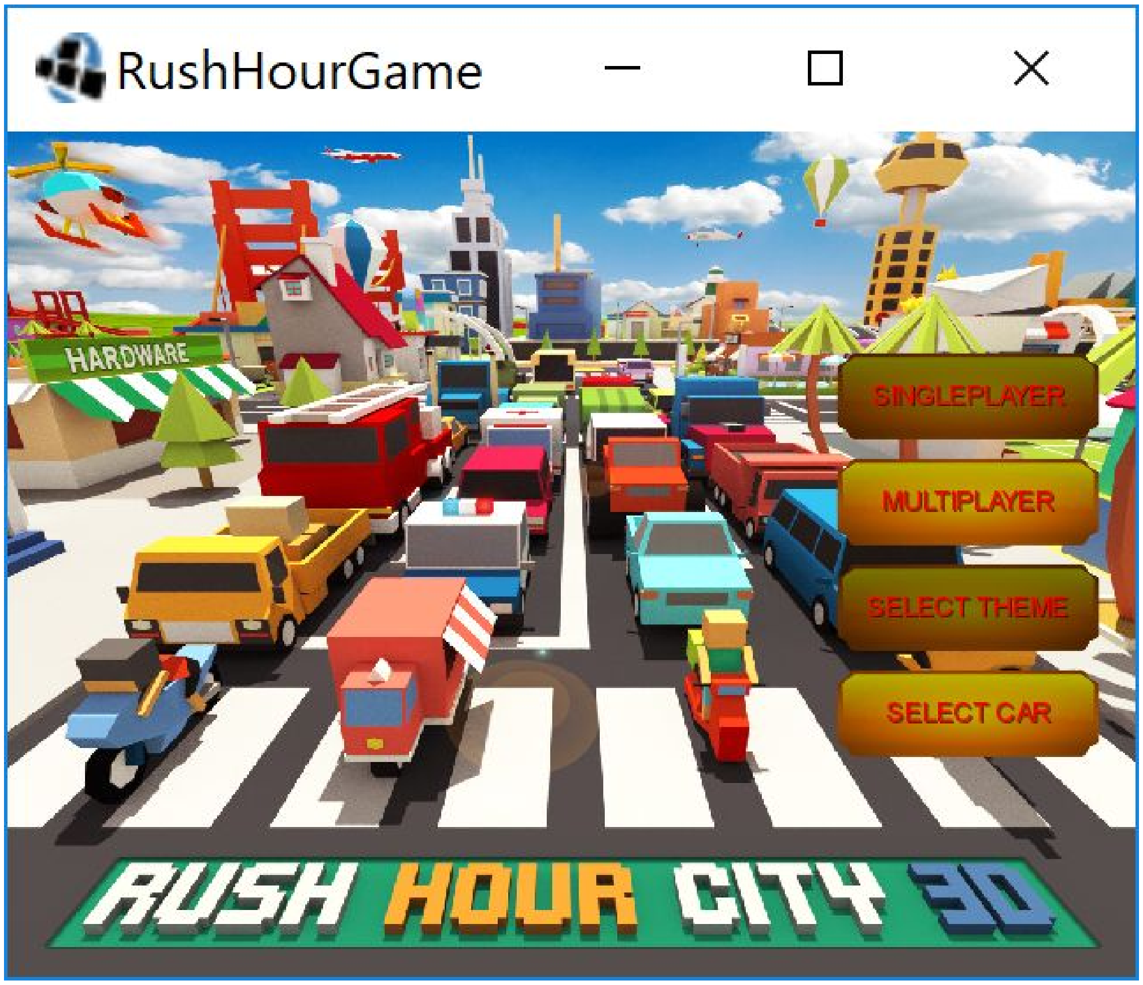
4.2 How to use

When the game launches, you see the main menu which you can choose options from. Currently, only the singleplayer mode is implemented, so click on the single player game mode. Choose the first map (which you are only eligible to) and start the game. You can mess around by clicking on a car and dragging it to the tile you want. Currently, there is no end to the game, so that’s it. You can quit the game by clicking the X button on the top.

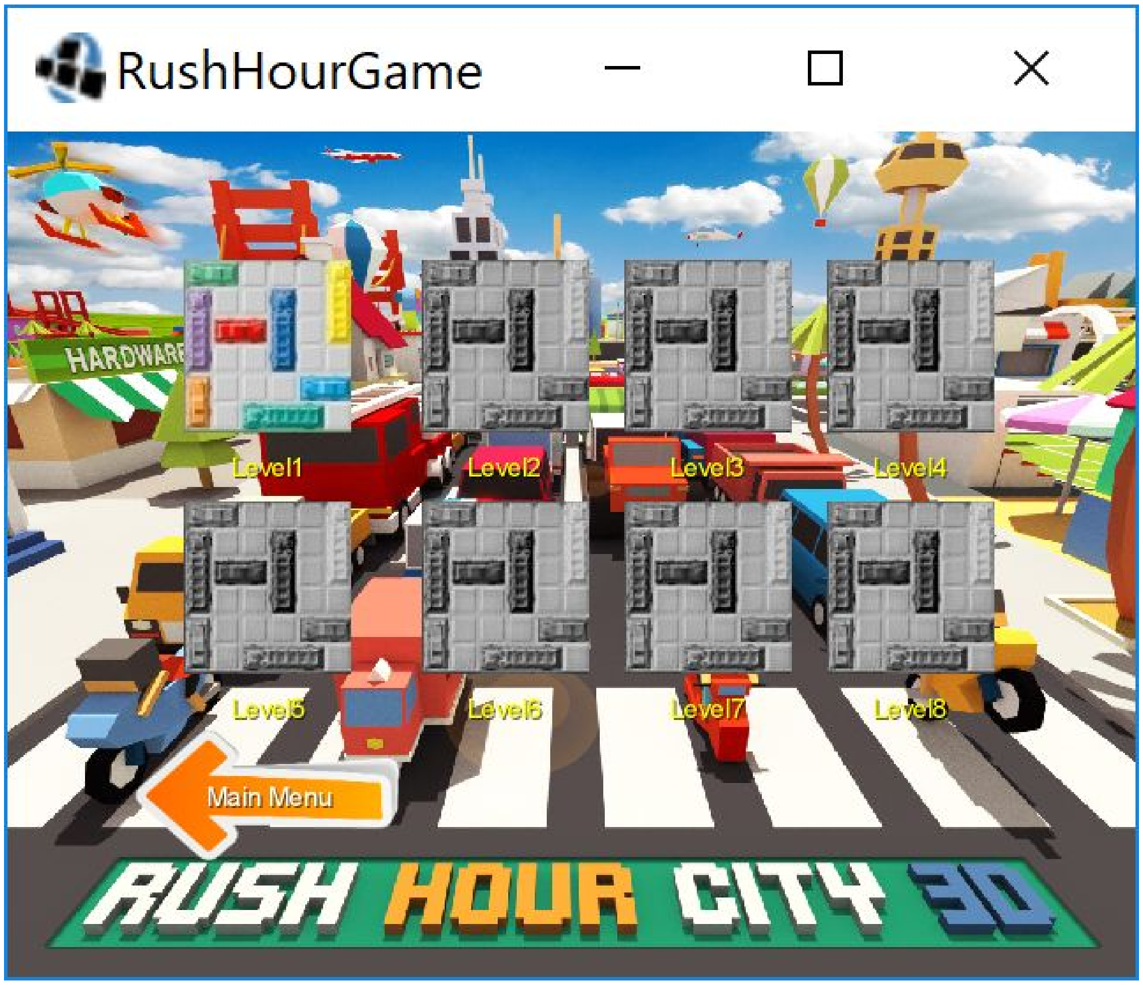
Appendices

Some screenshots from the game:

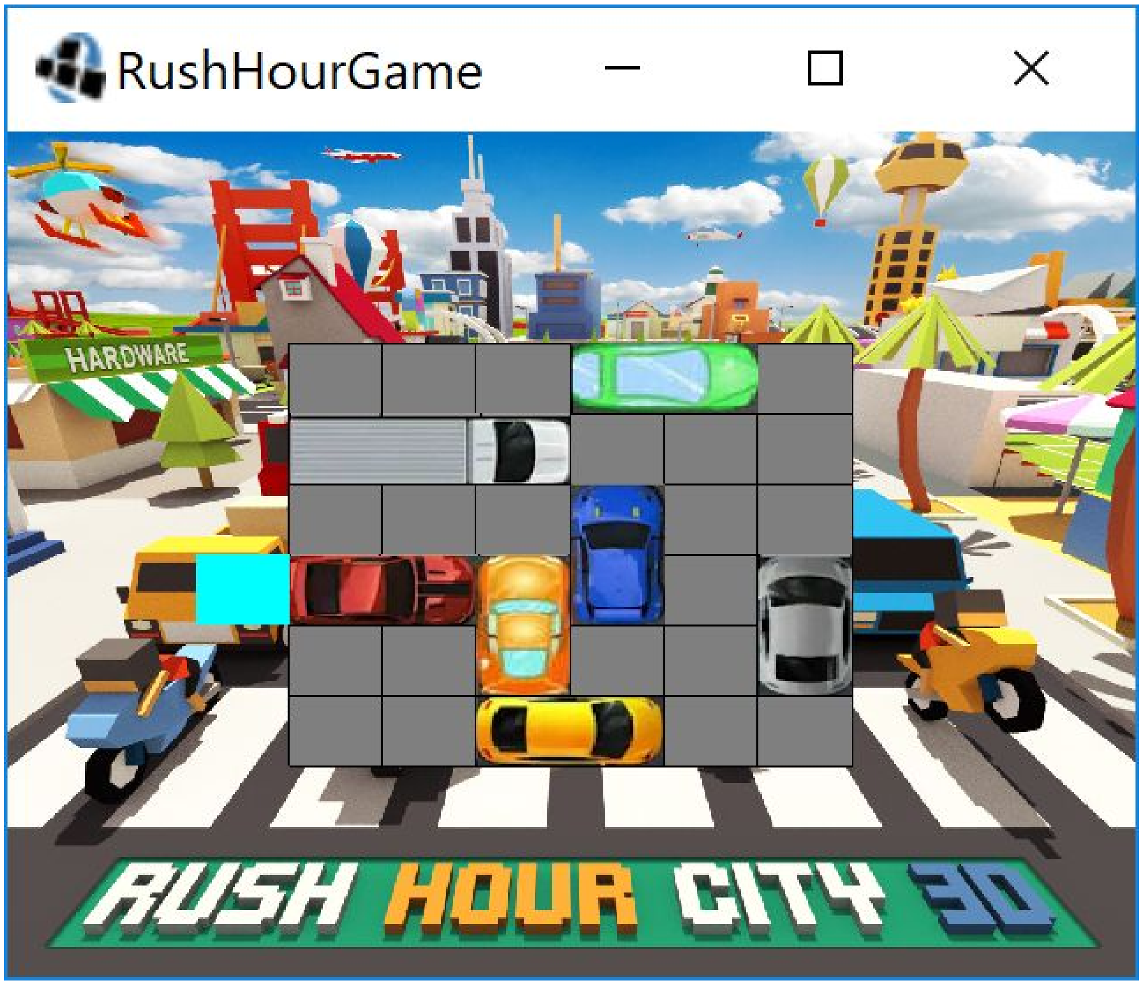
1-Main Menu: Shows up when the player opens the game.



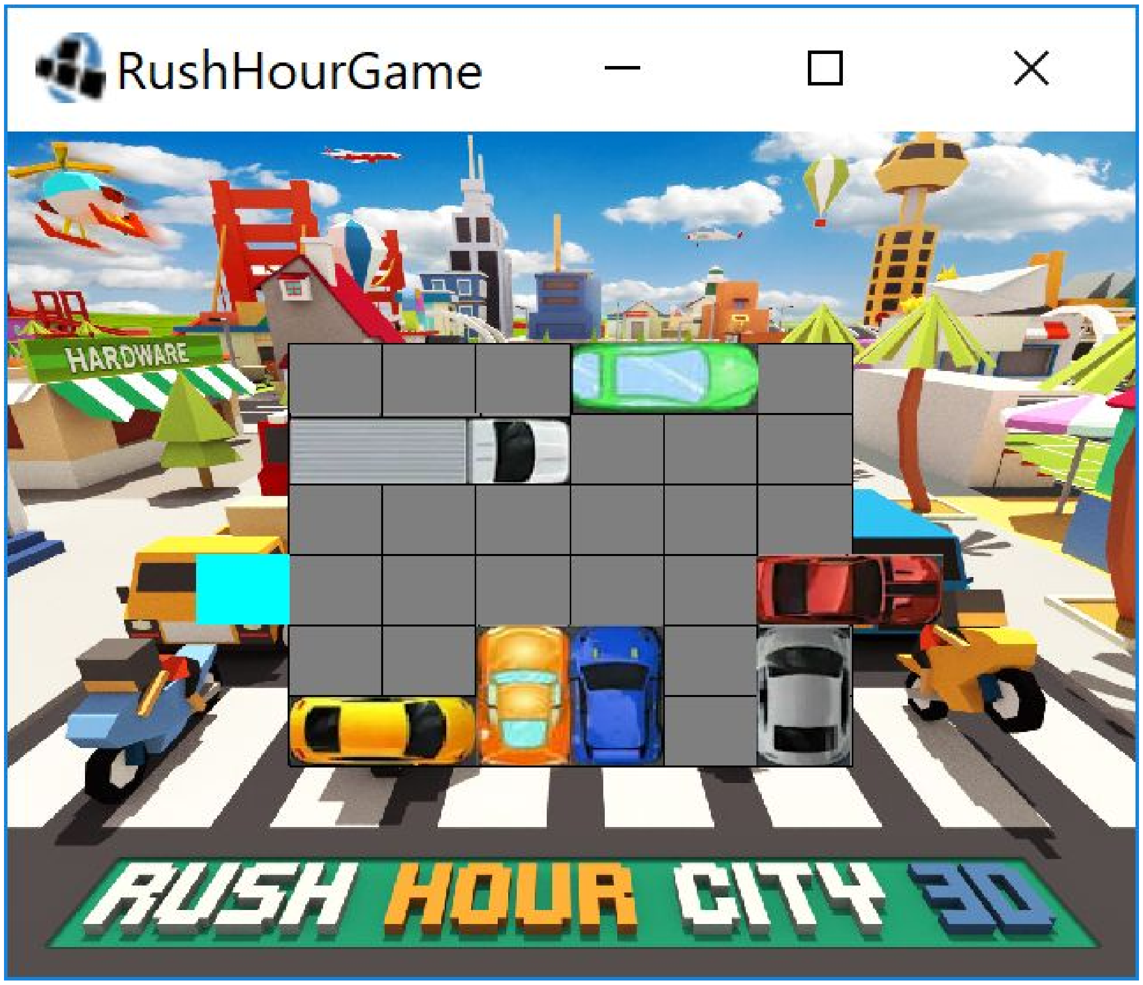
2-Map Selection for Single Player: Appears when the player chooses the singleplayer game mode. Since the player has to complete preceding levels to unlock the next one, **only the first level** is unlocked in the first place. However, unlocking other maps and other map arrangements is not applied at the moment



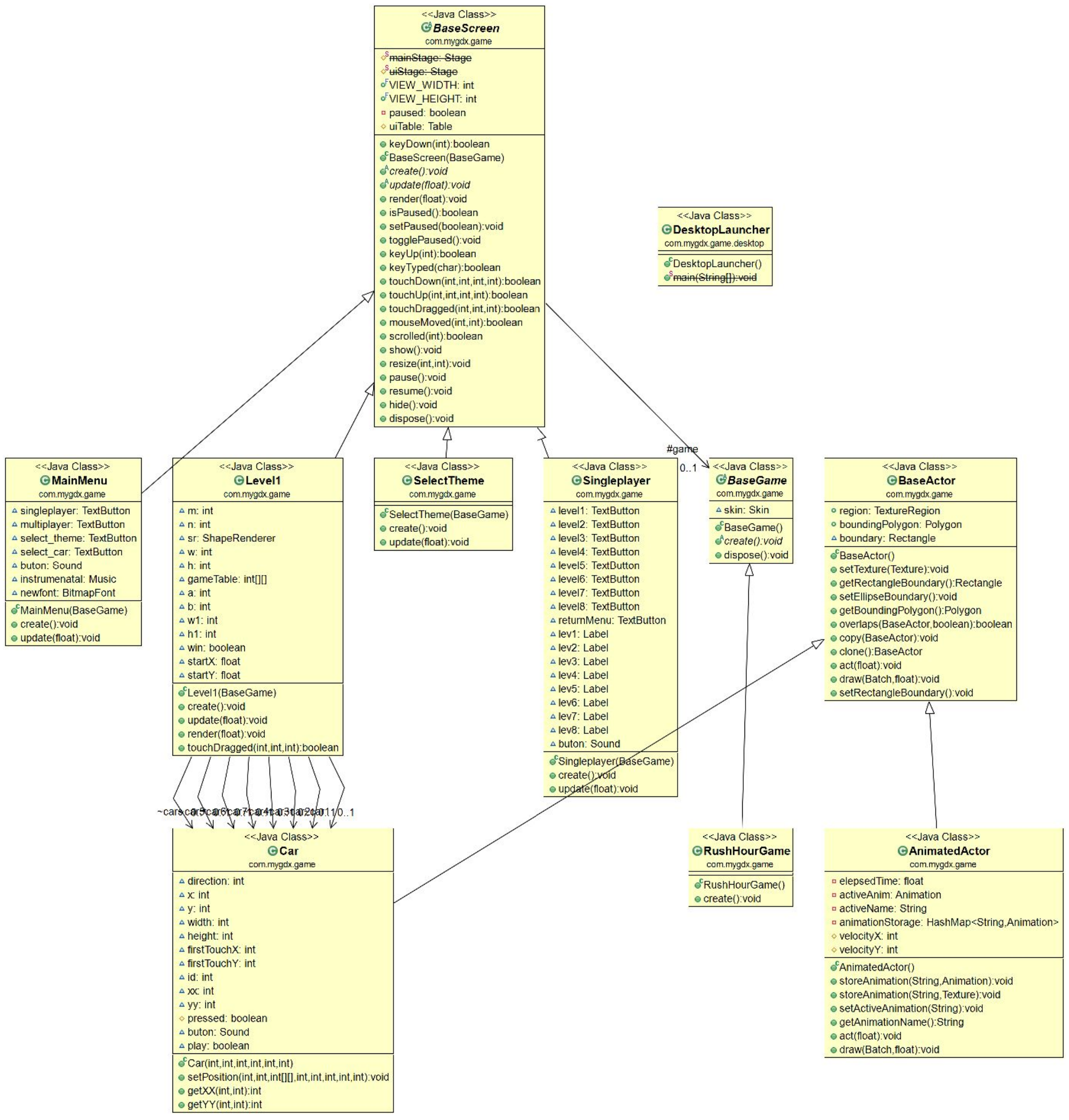
3-Singleplayer In-Game: When the player selects a level, the game starts and player moves cars to escape his car from the map. The initial arrangement of cars for **level 1** is shown below.



4- Singleplayer End-Level: When the user successfully escapes from the map successfully the current level ends. The car controlled by the user is shown as the red car in the picture below. After this step, the level is labeled as successful and the next level is going to be unlocked if present (Singleplayer campaign has not yet ended). The picture given below shows the end of **level 1**.



5-New Class Diagram: The class diagrams of our hitherto program is shown below. Majority of this changes were caused by the Game-Engine we are using since it makes our work easier this way. The diagram is structured using an Eclipse plugin, not Visual Paradigm.



<<Java Class>>
BaseScreen
com.mygdx.game
~~mainStage: Stage~~
~~uiStage: Stage~~
VIEW_WIDTH: int
VIEW_HEIGHT: int
paused: boolean
uiTable: Table
keyDown(int):boolean
BaseScreen(BaseGame)
create():void
update(float):void
render(float):void
isPaused():boolean
setPaused(boolean):void
togglePaused():void
keyUp(int):boolean
keyTyped(char):boolean
touchDown(int,int,int,int):boolean
touchUp(int,int,int,int):boolean
touchDragged(int,int,int):boolean
mouseMoved(int,int):boolean
scrolled(int):boolean
show():void
resize(int,int):void
pause():void
resume():void
hide():void
dispose():void
<<Java Class>>
DesktopLauncher
com.mygdx.game.desktop
DesktopLauncher()
~~main(String[]):void~~
#game
0..1
<<Java Class>>
MainMenu
com.mygdx.game
singleplayer: TextButton
multiplayer: TextButton
select_theme: TextButton
select_car: TextButton
buton: Sound
instrumenatal: Music
newfont: BitmapFont
MainMenu(BaseGame)
create():void
update(float):void
<<Java Class>>
Level1
com.mygdx.game
m: int
n: int
sr: ShapeRenderer
w: int
h: int
gameTable: int[][]
a: int
b: int
w1: int
h1: int
win: boolean
startX: float
startY: float
Level1(BaseGame)
create():void
update(float):void
render(float):void
touchDragged(int,int,int):boolean
<<Java Class>>
SelectTheme
com.mygdx.game
SelectTheme(BaseGame)
create():void
update(float):void
<<Java Class>>
Singleplayer
com.mygdx.game
level1: TextButton
level2: TextButton
level3: TextButton
level4: TextButton
level5: TextButton
level6: TextButton
level7: TextButton
level8: TextButton
returnMenu: TextButton
lev1: Label
lev2: Label
lev3: Label
lev4: Label
lev5: Label
lev6: Label
lev7: Label
lev8: Label
buton: Sound
Singleplayer(BaseGame)
create():void
update(float):void
<<Java Class>>
BaseGame
com.mygdx.game
skin: Skin
BaseGame()
create():void
dispose():void
<<Java Class>>
BaseActor
com.mygdx.game
region: TextureRegion
boundingPolygon: Polygon
boundary: Rectangle
BaseActor()
setTexture(Texture):void
getRectangleBoundary():Rectangle
setEllipseBoundary():void
getBoundingPolygon():Polygon
overlaps(BaseActor,boolean):boolean
copy(BaseActor):void
clone():BaseActor
act(float):void
draw(Batch,float):void
setRectangleBoundary():void
~cars car5 car6 car7 car4 car3 car2 car1 0..1
<<Java Class>>
Car
com.mygdx.game
direction: int
x: int
y: int
width: int
height: int
firstTouchX: int
firstTouchY: int
id: int
xx: int
yy: int
pressed: boolean
buton: Sound
play: boolean
Car(int,int,int,int,int,int)
setPosition(int,int,int[][],int,int,int,int,int):void
getXX(int,int):int
getYY(int,int):int
<<Java Class>>
RushHourGame
com.mygdx.game
RushHourGame()
create():void
<<Java Class>>
AnimatedActor
com.mygdx.game
elepsedTime: float
activeAnim: Animation
activeName: String
animationStorage: HashMap<String,Animation>
velocityX: int
velocityY: int
AnimatedActor()
storeAnimation(String,Animation):void
storeAnimation(String,Texture):void
setActiveAnimation(String):void
getAnimationName():String
act(float):void
draw(Batch,float):void