Architecture

Mozzarella Bytes | Team 18

Assessment N°2

Daniel Benison

Elizabeth Hodges

Kathryn Dale

Ravinder Dosanjh

Callum Marsden

Emilien Bevierre

Concrete UML architecture:

Diagram 1

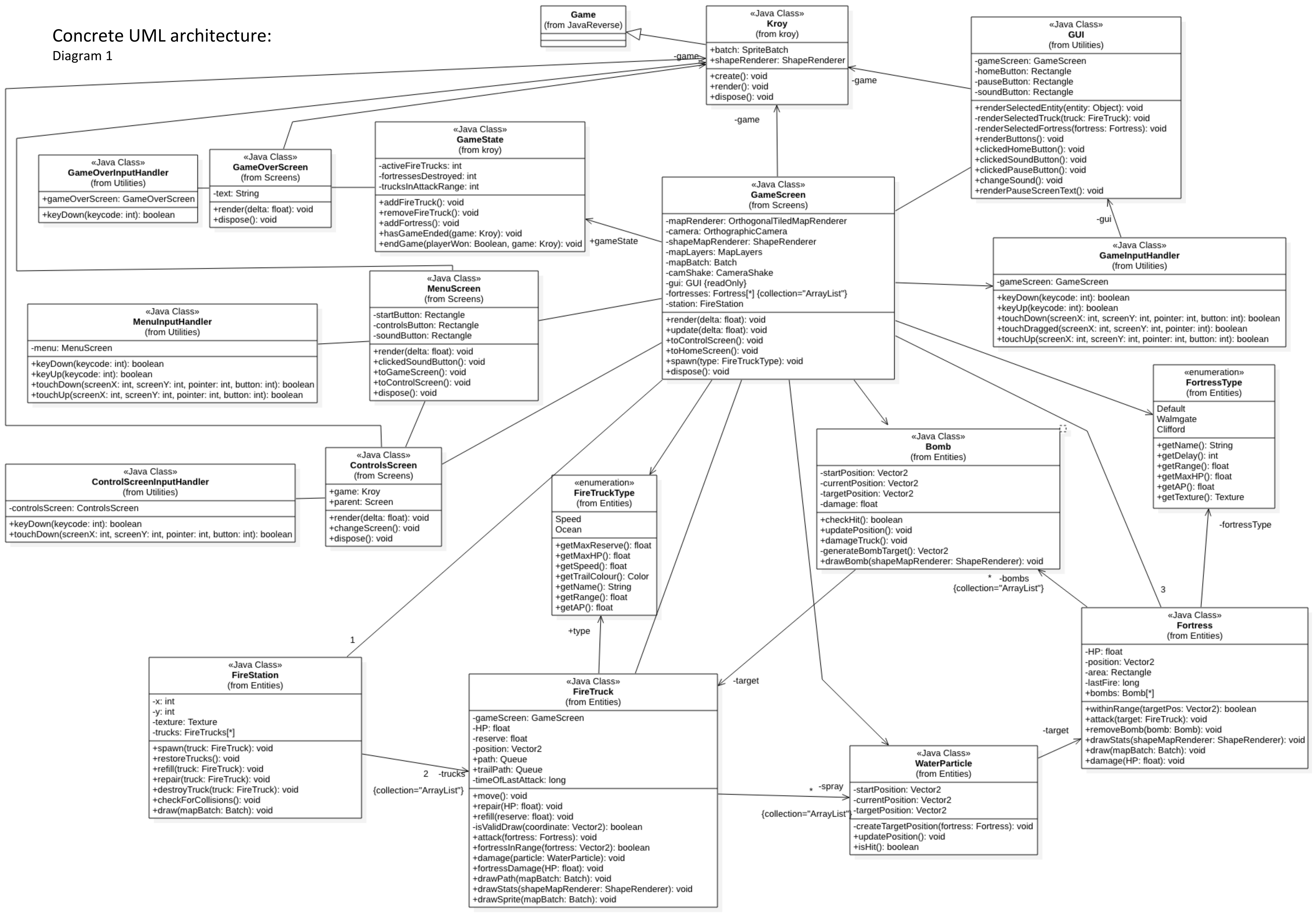


Diagram 1 shows the UML class diagram for our concrete architecture. We coded our game in Java (**FR_CODE**) using the LibGDX framework (this framework can be used on other devices (**FR_SCALABILITY**)). Diagram 1 was created using StarUML and a StarUML extension [1] to reverse Java code to UML. In diagram 1:

$$\begin{tikzcd}\text{Class A} \arrow[r] & \text{Class B}\end{tikzcd}$$

This arrow shows a directed association meaning A has unspecified navigability and B is navigable from A [2]

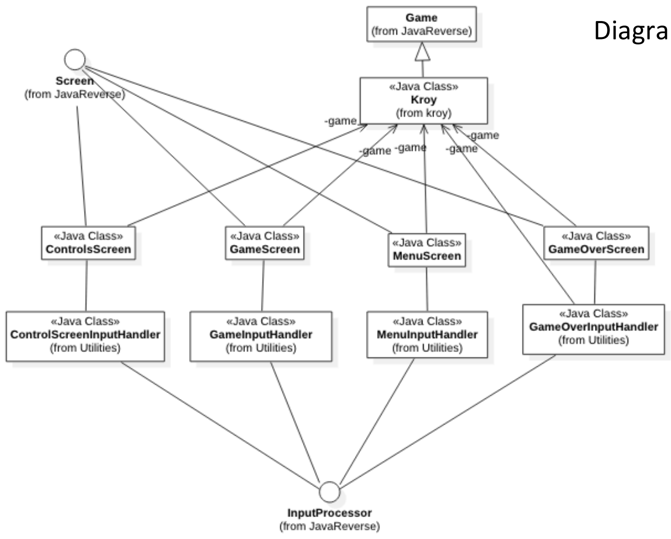
$$\begin{tikzcd}\text{Class C} \arrow[r, no head] & \text{Class D}\end{tikzcd}$$

This arrow shows an undirected association meaning both ends of the association have unspecified navigability [2]

$$\begin{tikzcd}\text{Class E} \arrow[r, -{Triangle[open]}] & \text{Class F}\end{tikzcd}$$

This arrow represents a generalisation where in OOP Class E inherits from Class F [3]

As seen in diagram 2, ControlsScreen (**FR_CONTROLS**), GameScreen, MenuScreen (**FR_MENU**) and GameOverScreen (**FR_GAME_OVER**) implement the LibGDX interface Screen, ControlScreenInputHandler, GameInputHandler, MenuInputHandler and GameOverInputHandler implement the LibGDX interface Application listener and Kroy extends the LibGDX class Game.



Game (from JavaReverse)
Screen (from JavaReverse)
«Java Class» Kroy (from kroy)
-game
-game
-game
-game
-game
«Java Class» ControlsScreen
«Java Class» GameScreen
«Java Class» MenuScreen
«Java Class» GameOverScreen
«Java Class» ControlScreenInputHandler (from Utilities)
«Java Class» GameInputHandler (from Utilities)
«Java Class» MenuInputHandler (from Utilities)
«Java Class» GameOverInputHandler (from Utilities)
InputProcessor (from JavaReverse)

Diagram 2

Diagram 3 shows a sequence diagram outlining the interaction between the screens. It was also created using StarUML.

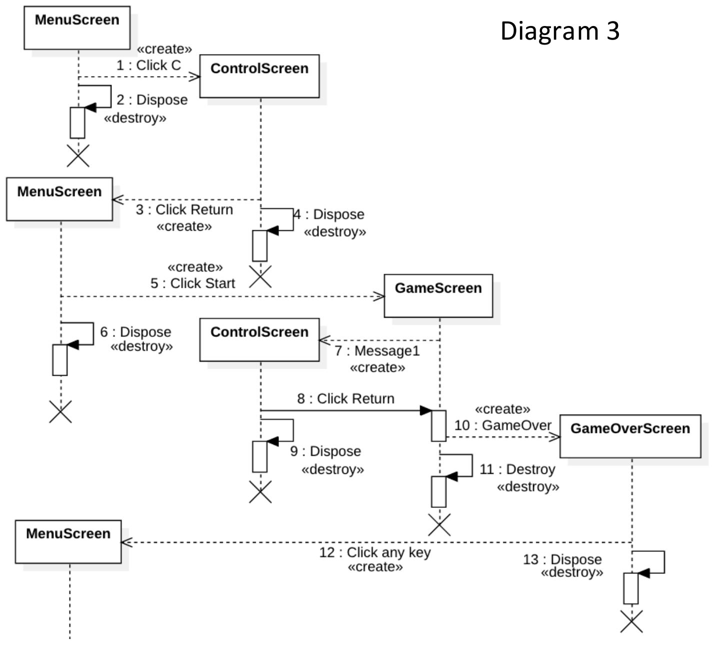


Diagram 3

Justification of concrete architecture

Using a tile map: This was so we could build a collision layer which we could use to check that a truck stayed on the road. It also meant that we could render the entire map once then only render the entities that changed each frame helping reduce lag (**NF_CONTROLS**)

Using touchdown(), touchDragged() and touchUp() in GameInputHandler: We decided the user should click on a truck and draw the route that the truck should follow (instead of using arrow keys or writing an algorithm for the trucks movement) so the user would have more control and these controls would work on other devices (**UR_SCALABILITY**). touchDown() is called when a user touches the screen and it checks a user has clicked on a truck, when the user "draws" a trucks path touchDragged is called which calls addTileToPath() in FireTruck adding the tile to the trucks trailPath if the player has drawn a continuous path on the road. Once the user stops touching the screen touchUp() is called which sets the moving to true so the truck starts to follows the path.

Path in FireTruck: GameInputHandler as described above adds the tiles that the player has drawn to into the trucks trailPath, however if each frame you rendered the truck to tiles present in trailPath it would look like the truck was jumping from tile to tile. Hence we used a Queue called path which contains intermediate values between each tile that the truck would be rendered to which makes it look like the truck is moving smoothly between tiles. A truck with slow speed generates more intermediate values between tiles hence it takes longer to move from tile to tile making it look slower.

TrailPath in FireTruck: Used in drawPath() to render a visual representation of the trucks path.

FireTruckType and FortressType enumerations: These were added so multiple objects of the same type could be created and they would not have to store the same values. Also, creating types with hard coded values makes it easier to check that the game is not too easy or too difficult whereas using randomly created values may make the game too easy in some instances and too hard in others.

Bomb class: This is how the fortress attacks the trucks (**FR_FORTRESS_ATTACK**). A fortress can have multiple bombs which it stores in an arrayList bomb. A bomb is automatically created by a fortress (**FR_AI**) once a truck is within the fortresses range and a predefined amount of time (delay in FortressType) has passed since the fortress attacked that truck. The bomb's start position is the fortress and its target position (generated in generateBombTarget()) is either the trucks position or a position near the truck, this adds an element of skills as the player has to avoid the bombs (**UR_ENJOYABILITY**).

WaterParticle Class: How a truck "floods" a fortress. While keydown() in GameInputHandler register that the user is pressing 'A' and a fortress is within the trucks range attack() in FireTruck is called each frame from GameScreen instantiating a Water Particle and adding it to the trucks spray if the truck's reserve > 0. Each render GameScreen calls WaterParticle isHit(),if a water particle hits a fortress fortressDamage() in FireTruck is called decreasing the fortresses HP by the trucks AP.

Input Handlers: Each screen has an input handler which implements the LibGDX interface InputProccessor (see diagram 2); this allows screens to responds to the player's input. We used the LibGDX interface because it is less likely to contain bugs then if we tried to implement it ourselves

Screens: Our game includes a menu (**FR_MENU**), game, controls (**FR_CONTROLS**) and game over screen (**FR_GAME_OVER**). Diagram 3 shows the interactions between screens. The only screen not disposed of when the screen changes is the game screen when the controls screen is clicked, this is so the user can resume their game when they exit the control screen.

How the concrete architecture builds from abstract architecture

Removed Entity class: In our abstract architecture, we had an Entity class containing an entity's location that Fortress, FireTruck, Station and Patrol inherited from. Instead we now pass the location of the entity into the entity's constructor from GameScreen as we decided there are not enough shared methods between the entity classes to warrant having an abstract entity class.

Replaced Abstract Screen class in abstract architecture with Screen interface from LibGDX: All screens use the Screen interface (see diagram 2) We decided to use an existing interface as it saves time and is less likely to contain bugs then if we implemented a solution ourselves.

Made ControlsScreen accessible from GameScreen: to increase **UR_PLAYABILITY** the control screen is now accessible from the game and menu screen; before it was only accessible from menu screen.

Added a Bomb class: Provides a visual way for the player to see the fortress is attacking the truck. Before damage() would have been called every frame causing continual damage to the truck whereas now a bomb is only generated after a predetermined number of seconds (different for each fortress) has passed and only causes damage if it lands in the same tile as a truck.

Added a GUI class: This renders and determines the actions for the control, pause, sound (**UR_SOUND_OFF**) and home button in the game screen.

Patrol Class and MiniGameScreen removed: Removed as they are not in required for assessment 2

Added InputHandlers, WaterParticle Class, FireTruckType and FortressType enumerations*:* See above for justifications

How concrete architecture implements the requirements

Requirements	How concrete architecture fulfils requirement
UR_WIN / UR_LOSE	When GameScreen calls spawn() the FireStation spawns a truck and calls addFireTrucks() which adds 1 to activeFireTrucks in GameState; when a fortress is destroyed GameScreen calls addFortress() which adds 1 to fortressesDestroyed in GameState. hasGameEnded (game: Kroy) is called each frame and calls endgame(boolean playerWon, game) with true if activeFireTrucks is 0 or false if fortress destroyed = 3, this triggers game over screen ending the game.
UR_REPAIR	Each render GameScreen calls restoreTrucks() in FireStation. If a truck is on one of its repair bays restoreTrucks calls refill (float), repair(float) in FireTruck which updates the trucks reserve and HP by the respective float amount.
UR_ FIRETRUCKS	The constructor of GameScreen calls spawn(FireTruckType) twice to instantiate 2 trucks. The spawn method increments activeFireTrucks in GameState and calls the spawn method in FireStation which adds the truck to its list of trucks.
UR_ FORTRESS	The constructor of GameScreen instantiates three fortresses, one from each FortressType and adds them to the arrayList of fortresses in GameScreen.
UR_PLAYER	All methods in the InputHandler classes can only deal with one input at a time meaning only one player can play.
UR_TRUCK_ SPACE	Each frame the update method in GameScreen calls checkForCollisions() in FireStation. This method iterates through the arrayList of trucks checking that the next tile in their trailPath is not occupied or going to be occupied by another truck. If there is a collision this method calls resetTrucks() which stops the trucks and moves them to adjacent tiles.
FR_FIRE_ TRUCK	Each FireTruck has a FireTruckType which contains a truck's: AP, max HP, max reserve, range, speed. There is one type for each truck.
FR_ FORTRESS	Each Fortress has a FortressType which contains a fortress': AP, max HP, range. There is one type for each fortress.
FR_ MOBILITY	A trucks movement is determined by input to GameInputHandler from the user. See previous page for further explanation.
FR_TRUCK_ ATTACK	A truck attacks a fortress while there is a fortress within the truck's range and the user is pressing 'A'. See previous page for further explanation.
FR_FORTRESS _ATTACK/ FR_AI	If a truck is within a fortresses range attack() in Fortress is called from GameScreen. This instantiates a bomb and adds it to the fortress' arrayList bombs. Each render GameScreen calls Bomb isHit(), if a bomb hits a fortress, damage() in Fortress is called from GameScreen decreasing the trucks HP by the fortress' AP.

References

[1] MK.Labs.Co, *StarUML Java extension,* Accessed on: 30.12.2019 [Online] Available at: http://staruml.io

[2] K. Fakhroutdinov, *UML association,* UML-diagrams.org, 2019, Accessed on: 27.12.2019. [Online] Available at: https://www.uml-diagrams.org/association.html

[3] M.Fowler, *UML Distilled Third Edition,* Massachusetts: Booch Jacob Rumbaugh, 2003. Accessed on: 27.12.2019. [Online] Available at: http://ce.sharif.edu/courses/96-97/2/ce418-1/resources/root/Books/UMLDistilled.pdf