SEPR 2019/20 Assessment 2

Team CheatCodez

Jonathan Grout, George Lesbirel, Cheuk Wang Wu, Lauren Quarshie, Muaz Atif, Lillian Coultas

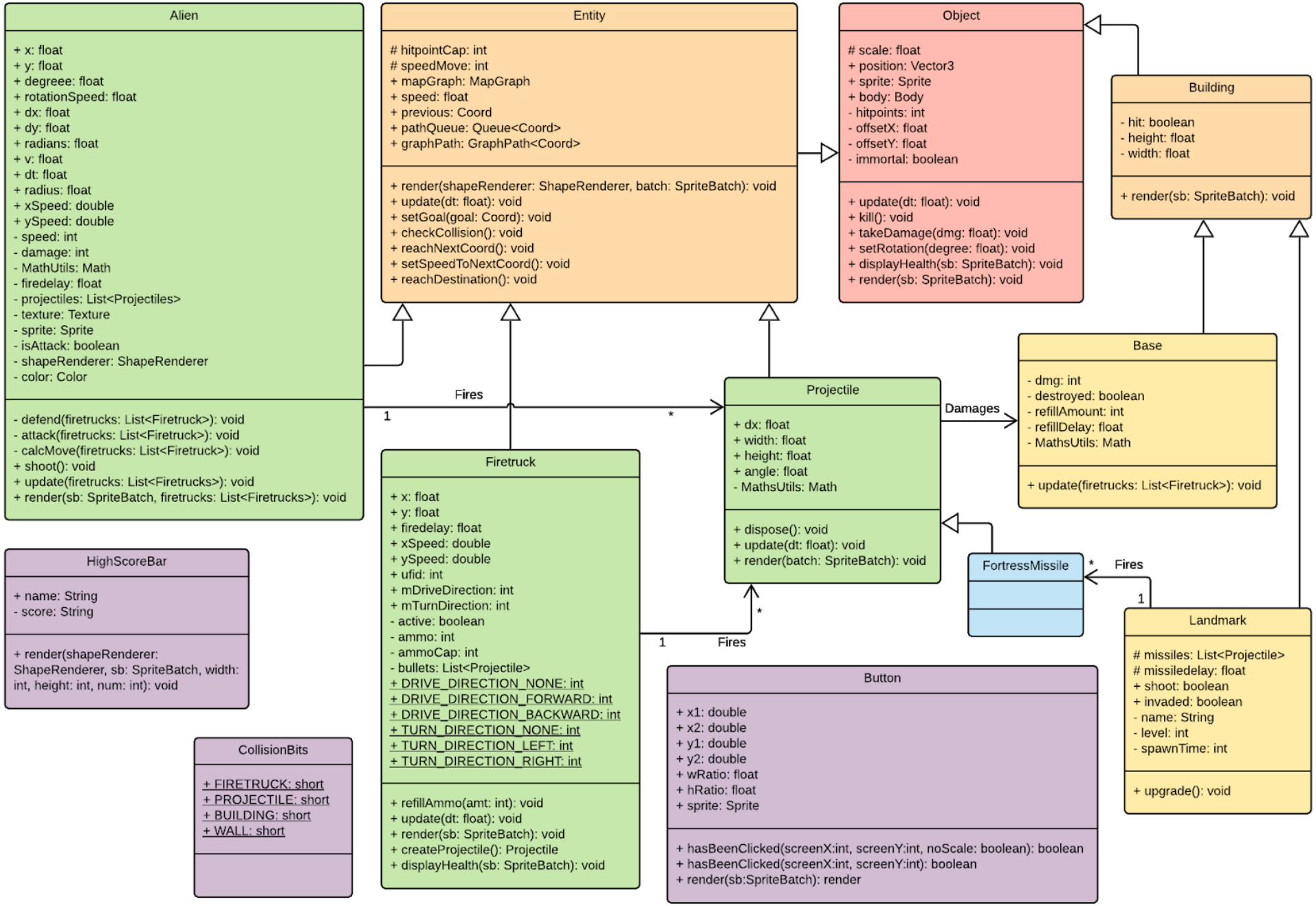
Architecture Report

The language used to describe the architecture is UML, with a diagram made using Lucidchart. UML was chosen as we felt it was the best way for us to model the structure of the system, as UML was designed to describe the structure of a system, but state diagrams and sequence diagrams would soon get very complicated due to how many operations they would have to include, and they are better suited to describing the behavior of a system, rather than the structure. Lucidchart was used to make the diagram because it allowed collaborators to help make the diagram, and also meant that it could be worked on from any computer without worrying and transferring files. Constructors, and getting/setting methods (methods which just return an attribute value or just set/update an attribute value) have not been included in the UML diagram to simplify it as much as possible. The UML diagram can also be found on the website (https://teamcheatcodez.github.io/project-kroy/supplements/concerete_uml.pdf)

We decided to use LibGDX as a graphics library for our game, as using a premade library would save us a lot of work, and be a lot more efficient. LibGDX has a lot of functionality, as well as being both widely used and open source, with good documentation. It works on android, desktop, and IOS with no emulators needed, making it a good choice for our game as in one of our client meetings for assessment one, it was mentioned that having a very portable game would be an advantage. Due to being a popular library, LibGDX has an active forum community, which was very useful when implementing the game, as it made it easy to get advice on implementation, or to diagnose bugs.

We also decided to use Box2D as a physics engine for our game. We chose Box2D because it was a very popular physics engine, again offering lots of community support for any errors or questions we had during implementation, and because it worked well with LibGDX. Using a physics engine for the game allowed us to implement collisions (i.e. for projectiles hitting objects) without having to add collision detection ourselves, and also made it easier to implement the firetruck movement, drastically reducing the amount of code we had to write ourselves, and therefore the amount of testing we had to do, allowing us to create a better quality game for the same amount of time spent on it.

There was quite a lot of change from the abstract architecture. Some of the changes were made in order to simplify the game, and remove superfluous classes - for example the classes “Waterblast”, “Projectile”, and “Laserblast” in the abstract architecture got changed during implementation, as just having one child for the Projectile class produced the same output with less unnecessary repetition and complication, making it more efficient. Some of the changes from the abstract architecture, like adding new classes, were mainly because we thought pre-made libraries would have the functionality they provided, and so didn’t include them in the abstract, then had to add them later. In the abstract architecture, we only planned to use LibGDX, and not Box2D, and this also accounts for some of the minor changes in classes, as Box2D required attributes or methods we hadn’t planned for. Changes to the abstract architecture is detailed and justified in the table below the UML diagram.


Alien
+ x: float
+ y: float
+ degreee: float
+ rotationSpeed: float
+ dx: float
+ dy: float
+ radians: float
+ v: float
+ dt: float
+ radius: float
+ xSpeed: double
+ ySpeed: double
- speed: int
- damage: int
- MathUtils: Math
- firedelay: float
- projectiles: List<Projectiles>
- texture: Texture
- sprite: Sprite
- isAttack: boolean
- shapeRenderer: ShapeRenderer
- color: Color
- defend(firetrucks: List<Firetruck>): void
- attack(firetrucks: List<Firetruck>): void
- calcMove(firetrucks: List<Firetruck>): void
+ shoot(): void
+ update(firetrucks: List<Firetrucks>): void
+ render(sb: SpriteBatch, firetrucks: List<Firetrucks>): void
Entity
hitpointCap: int
speedMove: int
+ mapGraph: MapGraph
+ speed: float
+ previous: Coord
+ pathQueue: Queue<Coord>
+ graphPath: GraphPath<Coord>
+ render(shapeRenderer: ShapeRenderer, batch: SpriteBatch): void
+ update(dt: float): void
+ setGoal(goal: Coord): void
+ checkCollision(): void
+ reachNextCoord(): void
+ setSpeedToNextCoord(): void
+ reachDestination(): void
Object
scale: float
+ position: Vector3
+ sprite: Sprite
+ body: Body
- hitpoints: int
- offsetX: float
- offsetY: float
- immortal: boolean
+ update(dt: float): void
+ kill(): void
+ takeDamage(dmg: float): void
+ setRotation(degree: float): void
+ displayHealth(sb: SpriteBatch): void
+ render(sb: SpriteBatch): void
Building
- hit: boolean
- height: float
- width: float
+ render(sb: SpriteBatch): void
Base
- dmg: int
- destroyed: boolean
- refillAmount: int
- refillDelay: float
- MathsUtils: Math
+ update(firetrucks: List<Firetruck>): void
Projectile
+ dx: float
+ width: float
+ height: float
+ angle: float
- MathsUtils: Math
+ dispose(): void
+ update(dt: float): void
+ render(batch: SpriteBatch): void
Damages
Fires
1
*
Firetruck
+ x: float
+ y: float
+ firedelay: float
+ xSpeed: double
+ ySpeed: double
+ ufid: int
+ mDriveDirection: int
+ mTurnDirection: int
- active: boolean
- ammo: int
- ammoCap: int
- bullets: List<Projectile>
+ DRIVE_DIRECTION_NONE: int
+ DRIVE_DIRECTION_FORWARD: int
+ DRIVE_DIRECTION_BACKWARD: int
+ TURN_DIRECTION_NONE: int
+ TURN_DIRECTION_LEFT: int
+ TURN_DIRECTION_RIGHT: int
+ refillAmmo(amt: int): void
+ update(dt: float): void
+ render(sb: SpriteBatch): void
+ createProjectile(): Projectile
+ displayHealth(sb: SpriteBatch): void
1
Fires
*
FortressMissile
*
Fires
1
Landmark
missiles: List<Projectile>
missiledelay: float
+ shoot: boolean
+ invaded: boolean
- name: String
- level: int
- spawnTime: int
+ upgrade(): void
HighScoreBar
+ name: String
- score: String
+ render(shapeRenderer: ShapeRenderer, sb: SpriteBatch, width: int, height: int, num: int): void
CollisionBits
+ FIRETRUCK: short
+ PROJECTILE: short
+ BUILDING: short
+ WALL: short
Button
+ x1: double
+ x2: double
+ y1: double
+ y2: double
+ wRatio: float
+ hRatio: float
+ sprite: Sprite
+ hasBeenClicked(screenX:int, screenY:int, noScale: boolean): boolean
+ hasBeenClicked(screenX:int, screenY:int): boolean
+ render(sb:SpriteBatch): render

Justification

Software Component	Changes from the Abstract Architecture	Justification	Relevant Requirements
Object	No significant changes.	The Object class is the base class for most of the game, this is efficient and removes duplication as it can hold all the methods and attributes most of the objects in the game will require (such as a sprite attribute, or a render method - mostly concerned with displaying the object) without having to put them in every class.	
Entity	No significant changes.	Entity, like Object, holds methods and attributes that Firetruck, Alien, and Projectile have in common (mostly concerned with the movement of the entity). These are not in Object as Buildings don't require them, only Entities - and this also ensures Object doesn't become a 'God Class' with too much functionality.	FR_ENGINE-SPECS, UR_ENGINE_CONTROL
Building	No significant changes.	Building is the parent class for Landmark and Base, and so holds attributes and methods that they will both need. It contains methods specific to Buildings taking damage, as Object and Entity don't require these, or the movement.	UR_FORTRESS_NO, UR_SPECS-FORTRESS, FR_FORTRESS_SPECS
Button	New class, not in abstract architecture.	The Button class was added to be used for anything clickable, like the switch between Firetrucks, or the main menu buttons. It was added outside of the abstract architecture as we didn't know it would be necessary, we thought that a library would provide the functionality we needed.	
CollisionBits	New class, not in abstract architecture.	CollisionBits stores the collision bits, which are used to make sure the collisions can be detected, and also the objects don't clip through other objects, so we can tell that Firetrucks have been hit, but they can't drive through walls. It was also added outside of the abstract architecture.	UR_ENGINE_CONTROL
Base	No significant changes.	Base has no changes from the abstract architecture beyond minor ones needed to implement refilling the water in the Firetrucks, and healing them.	UR_REPAIR, UR_REFILL
Firetruck	No significant changes.	Firetruck requires more attributes than are shown in abstract architecture because of how	UR_ENGINE_SPECS,

		movement was implemented, using Box2D, but no major changes were implemented.	FR_ENGINE_SPECS
HighScore Bar	New class, not in abstract architecture.	This is used to display the high scores to the user from the main menu. It was also added outside of the abstract architecture as we didn't know it would be necessary.	UR_COMPETITIVE, FR_PLAYER_NAME
Landmark	Landmark is a merging of "Alienbase" and "Landmark" from the abstract architecture.	In the abstract architecture, Building was split into three classes, "Base", "Alienbase", and "Landmark". This was an unnecessary separation with "Alienbase" and "Landmark" being very similar with similar functionality, so in order to have fewer arbitrary classes, "Alienbase" and "Landmark" were merged into only one class with the same functionality.	UR_FORTRESS_NO, FR_FORTRESS_IMPROVEMENT
Alien	Alien has no significant changes, however the UFO class was removed.	There was no need to add complications and extras like having UFOs as well as Aliens, so between this and time constraints, Alien is now the only enemy for the player. This also makes the game simpler from the player's point of view.	UR_ET_PATROLS, FR_PATROLS, UR_ET_WEAPONS
Projectile	Projectile is a merging of "Projectile" and "Laserblast" from the abstract architecture.	In the abstract architecture, projectile was split into two further classes, "Waterblast", and "Laserblast". This was an unnecessary separation, as above with "Landmark" and "Alienbase", so in order to have fewer arbitrary classes, "Laserblast", "Waterblast", and "Projectile" were merged into only two classes with the same functionality.	UR_ENGINE_WEAPONS
FortressMissile	FortressMissile is built from "Waterblast" from the abstract architecture.	As above, FortressMissile is like "Waterblast", but with a few minor changes due to Projectile having increased functionality. The name was also changed for clarity, and to make collision detection easier to implement.	UR_ET_WEAPONS
Stage	Removed.	Stages were removed from the architecture as, instead of splitting the game model into nine small sections, we decided having one large section was simpler, and more efficient to implement. This made "Stage" no longer necessary.	N/A
Map	Removed.	We removed "Map" in favour of the LibGDX map classes and tools, as using them instead of making our own reduced the amount of work we had to do to implement the game.	N/A