Group-19 Phase 2

Design and Implementation

Our overall approach to implementing our game was to build it from a bottom-up approach using the selected framework libGDX. Many members of our team have worked with OpenGL, before so choosing libGDX, a framework which wraps OpenGL, was a reasonable decision for us to move forward with. During phase one, our team had already designed a part of our game regarding requirements and features. At the start of phase two, we wanted to expand on our discussions from phase one and completely design our game fundamentals before starting the implementation process. Once our game design was done, we began to build a runnable desktop application to push onto our master repository for everyone to set up their local repositories and environments. Subsequently, we divided the roles and each member would implement their required responsibilities. We decided to add the basic functionalities first and then build our way up towards the more time consuming functions. For example, movement of a player going up, down, left, right is relatively simple compared to the functionality of a player colliding with an item. Whenever a member would be done with a certain functionality, they would merge their local repository into master to make sure that other members can keep up with the fast pace changes.

Another aspect of our game is the use of sprites for artwork. Our sprites were all downloaded from a website known as The Spriters Resource (The Spriters Resource**)** which holds a collection of resources such as sprites for personal projects and non-commercial work. Using the The Spriters Resource website we were able to pull sprites that matched our game theme and make animations for players and enemies such as shoot, hurt, or run animations.

Management Process

Our management process consisted of weekly and sometimes biweekly meetings in person and over VoIP. During these meetings, we would assign features to each member to branch and implement before our next meeting. The next meeting would then start with us reviewing the newly merged build before assigning new features to work on. During these meetings, we would also review the state of the game to determine if we should keep, remove, or implement new features that would make our game unique.

In our first meeting for phase two, we briefly divided the roles in order to get a template of the game running. To specify, we each began working on a different class such as the player, enemy, map and items, and menus with a deadline assigned in order to make sure we were managing our time wisely. Every week, during our in person or VoIP meetings, we would review the current state of the project, determine what requirements are still required, raise awareness to bugs, and redistribute responsibilities. For the most part, each individual continued adding functionality to the branches they were already working on. For example, Caleb who was

working on the player class began by making sure the player had the basic movements. As the weeks went by Caleb would add more functionality and logic to the player such as collision, obtaining items, and attacking.

External Libraries

We decided to implement the game using the libGDX framework (libGDX) which also wraps OpenGL. The framework takes care of some file I/O which would have been boilerplate to write. We also chose it due to it's wrapping of OpenGL which makes rendering to a window a more straightforward process.

Code Quality

When going from the design process to implementation, we found that in order to keep our code quality we would have to make a few changes to our initial design and UML diagram. The changes include making a new Tile class which extends Entity and making the Rewards and Punishments extend Tile. This allowed us to have a cleaner representation of the Map by splitting it into Tiles. We removed the MoveableEntity class and moved its children to just Entity as there were barely any differences between the two so it wasn't necessary to have a whole separate class. To add to these changes, we also modified the names of some of our classes to add further clarity to our code. GameBoard → Map and HealthReward → Food are just a few examples where we found this improved code readability.

Another improvement of quality we made on our code was to make our algorithms more efficient in speed and memory. For our first iteration of our path finding algorithm we used the concept of depth first search. However, when adding multiple enemies to the game level, the recursive trees of each enemy would be so deep that our whole game started having input and response delays. As a result we decided to use the Bellman-Ford algorithm, a relatively simple algorithm to implement as compared to A-star search which would require us to test multiple heuristics. Under the Bellman-Ford algorithm, our game now runs much more smoothly as compared to previous iterations.

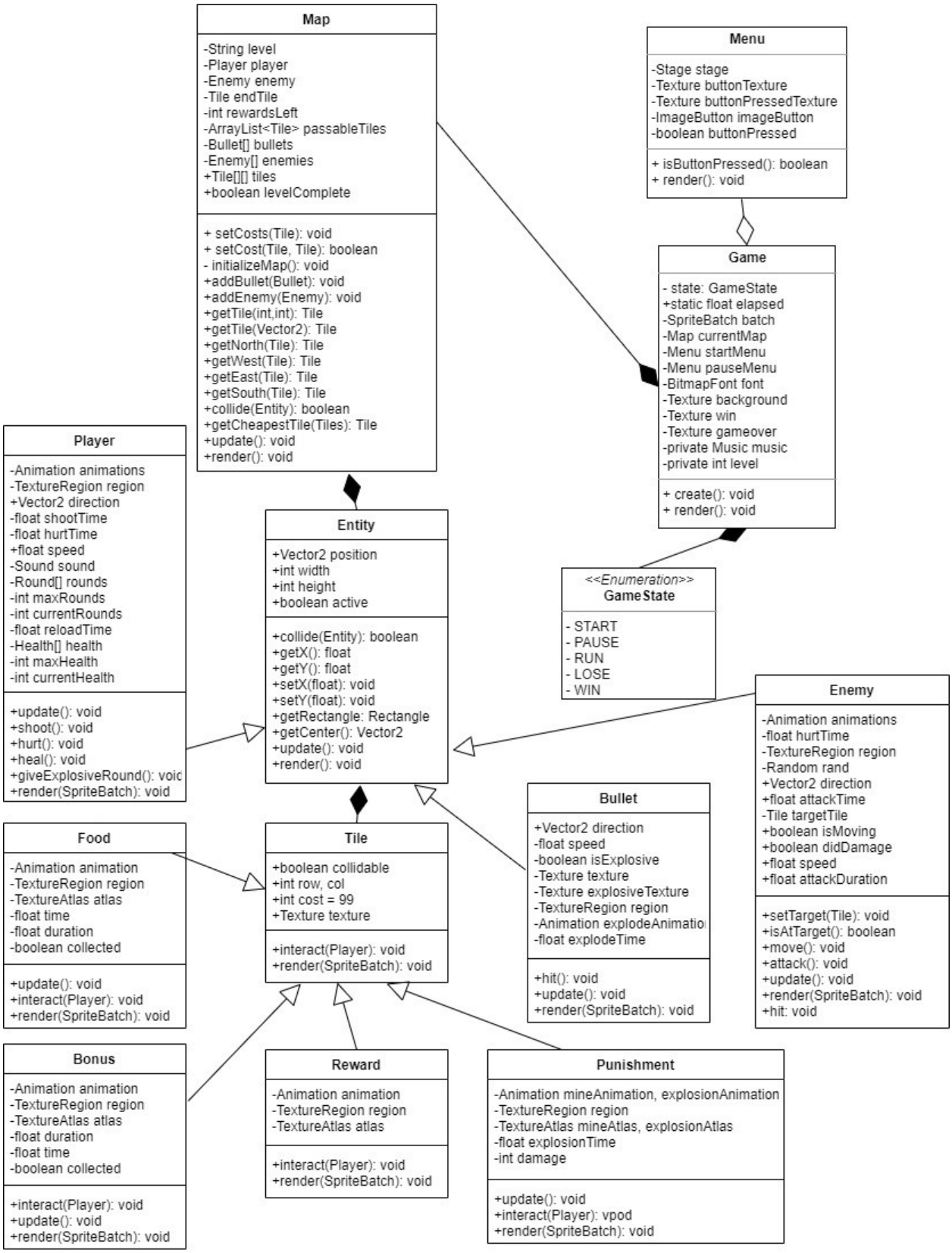
Challenges

The challenges we faced during this phase can be placed into two categories, logistic and coding. The logistic challenges we faced mainly revolved around scheduling in-person meetings as we all had different conflicting class schedules. As the end of this phase was closing in, classes were abruptly cancelled which meant our last meetings before the due date would have to be rescheduled over VoIP.

When it comes to coding challenges, one of the bigger challenges was writing effective JavaDoc comments. When reading another team member's code, it was important to be able to fully understand it at first glance so that it could be built upon without conflict of not knowing what the code segment actually does. Another obstacle was writing effective and relevant

variable names. Even if a comment can explain a code segment, the code segment itself needs to be readable. An example of this would be changing our level completed flag from levelFlag → levelComplete. With the new version, one can immediately identify the variable is a flag *and* the flag's purpose. We found that keeping this level of code quality actually sped up the development process as there was less confusion and misunderstandings.

(On the next page is an updated version of our UML diagram to reflect phase 2's modification)


Map
-String level
-Player player
-Enemy enemy
-Tile endTile
-int rewardsLeft
-ArrayList<Tile> passableTiles
-Bullet[] bullets
-Enemy[] enemies
+Tile[][] tiles
+boolean levelComplete
+ setCosts(Tile): void
+ setCost(Tile, Tile): boolean
- initializeMap(): void
+addBullet(Bullet): void
+addEnemy(Enemy): void
+getTile(int,int): Tile
+getTile(Vector2): Tile
+getNorth(Tile): Tile
+getWest(Tile): Tile
+getEast(Tile): Tile
+getSouth(Tile): Tile
+collide(Entity): boolean
+getCheapestTile(Tiles): Tile
+update(): void
+render(): void
Menu
-Stage stage
-Texture buttonTexture
-Texture buttonPressedTexture
-ImageButton imageButton
-boolean buttonPressed
+ isButtonPressed(): boolean
+ render(): void
Game
- state: GameState
+static float elapsed
-SpriteBatch batch
-Map currentMap
-Menu startMenu
-Menu pauseMenu
-BitmapFont font
-Texture background
-Texture win
-Texture gameover
-private Music music
-private int level
+ create(): void
+ render(): void
<<Enumeration>>
GameState
- START
- PAUSE
- RUN
- LOSE
- WIN
Player
-Animation animations
-TextureRegion region
+Vector2 direction
-float shootTime
-float hurtTime
+float speed
-Sound sound
-Round[] rounds
-int maxRounds
-int currentRounds
-float reloadTime
-Health[] health
-int maxHealth
-int currentHealth
+update(): void
+shoot(): void
+hurt(): void
+heal(): void
+giveExplosiveRound(): voic
+render(SpriteBatch): void
Entity
+Vector2 position
+int width
+int height
+boolean active
+collide(Entity): boolean
+getX(): float
+getY(): float
+setX(float): void
+setY(float): void
+getRectangle: Rectangle
+getCenter(): Vector2
+update(): void
+render(): void
Enemy
-Animation animations
-float hurtTime
-TextureRegion region
-Random rand
+Vector2 direction
+float attackTime
-Tile targetTile
+boolean isMoving
+boolean didDamage
+float speed
+float attackDuration
+setTarget(Tile): void
+isAtTarget(): boolean
+move(): void
+attack(): void
+update(): void
+render(SpriteBatch): void
+hit: void
Food
-Animation animation
-TextureRegion region
-TextureAtlas atlas
-float time
-float duration
-boolean collected
+update(): void
+interact(Player): void
+render(SpriteBatch): void
Tile
+boolean collidable
+int row, col
+int cost = 99
+Texture texture
+interact(Player): void
+render(SpriteBatch): void
Bullet
+Vector2 direction
-float speed
-boolean isExplosive
-Texture texture
-Texture explosiveTexture
-TextureRegion region
-Animation explodeAnimatio
-float explodeTime
+hit(): void
+update(): void
+render(SpriteBatch): void
Bonus
-Animation animation
-TextureRegion region
-TextureAtlas atlas
-float duration
-float time
-boolean collected
+interact(Player): void
+update(): void
+render(SpriteBatch): void
Reward
-Animation animation
-TextureRegion region
-TextureAtlas atlas
+interact(Player): void
+render(SpriteBatch): void
Punishment
-Animation mineAnimation, explosionAnimation
-TextureRegion region
-TextureAtlas mineAtlas, explosionAtlas
-float explosionTime
-int damage
+update(): void
+interact(Player): vpod
+render(SpriteBatch): void