Computer Science Department

San Francisco State University

CSC 413 Spring 2018

Term Project Report

Github repository:
https://github.com/sfsu-csc-413-spring-2018/term-project-team-b

Compiling Instructions: -

Tank Game
javac src/tanks/Main.java
java src/tanks/Main

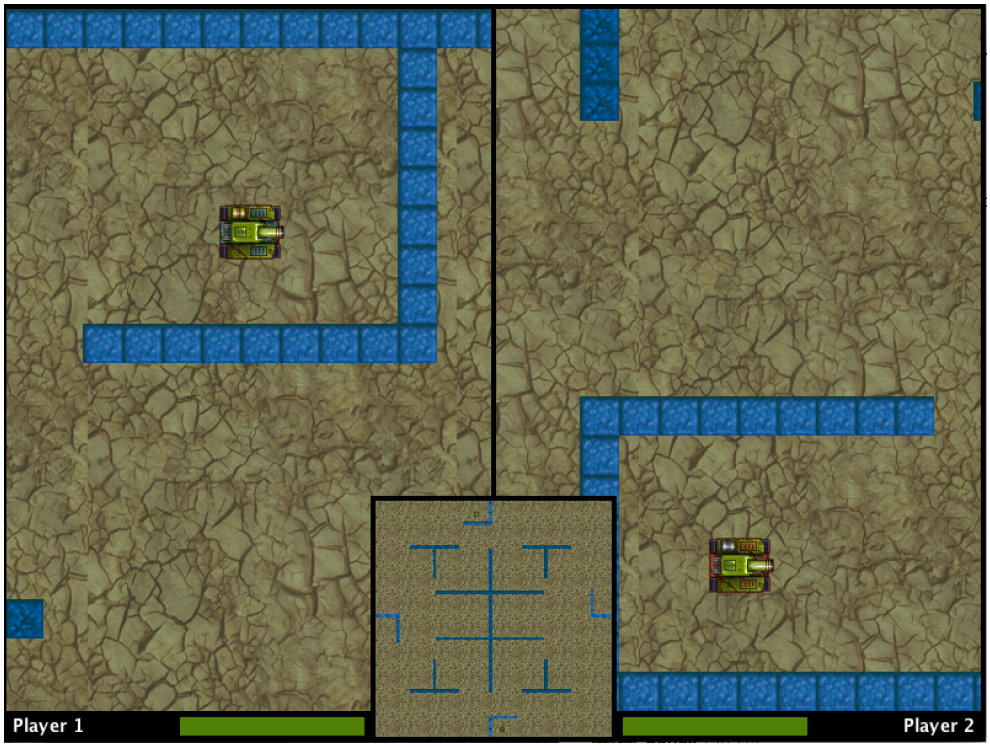
Galactic Mail
javac src/galactic/Main.java
java src/galactic/Main

Name	Student ID
Gayatri Pise	917922296
Soheil Ansari	917951156

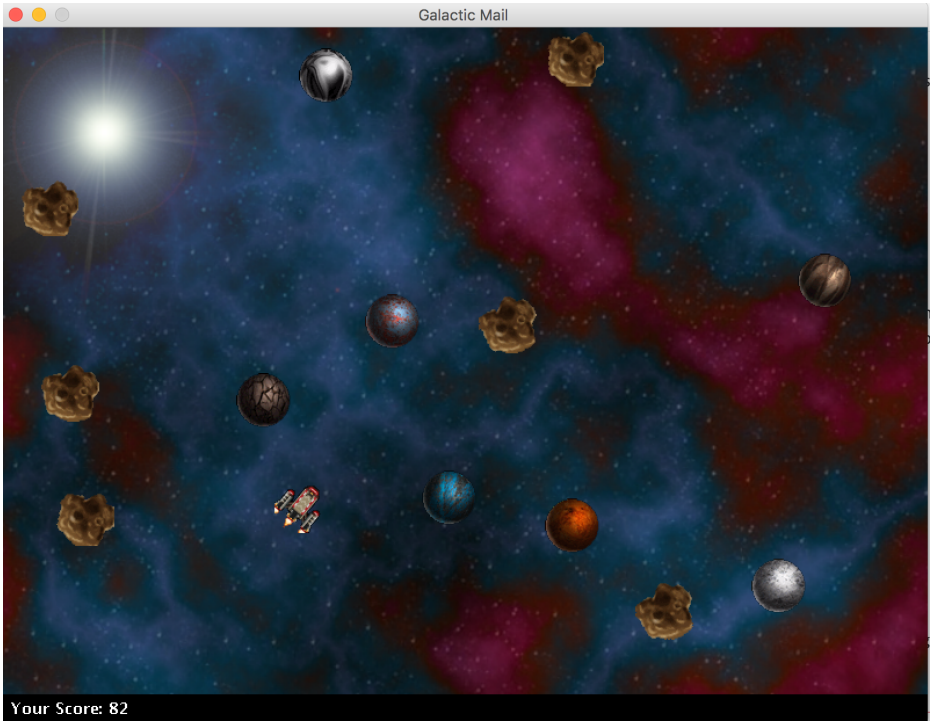
1. Project Introduction

As part of the term project, we have to create an object-oriented solution for a Tank Game and Galactic Mail game. The solution should be designed such that both the games reuse components from one another. Both the games use Java GUI and graphics components.

The Tank Game is a two-player game where the objective of any player is to damage the other player's tank. The tanks can fire bullets in any direction and the bullet continues to travel until it collides with any obstruction. The tank life decreases on every collision with the bullet. In addition to the tanks, the game also consists of destructible and indestructible walls. On a bullet collision, the destructible wall is destroyed and there is no damage to the indestructible wall. A player wins when there is no more life left in the other tank. Both the players use the computer keys to maneuver the tank.

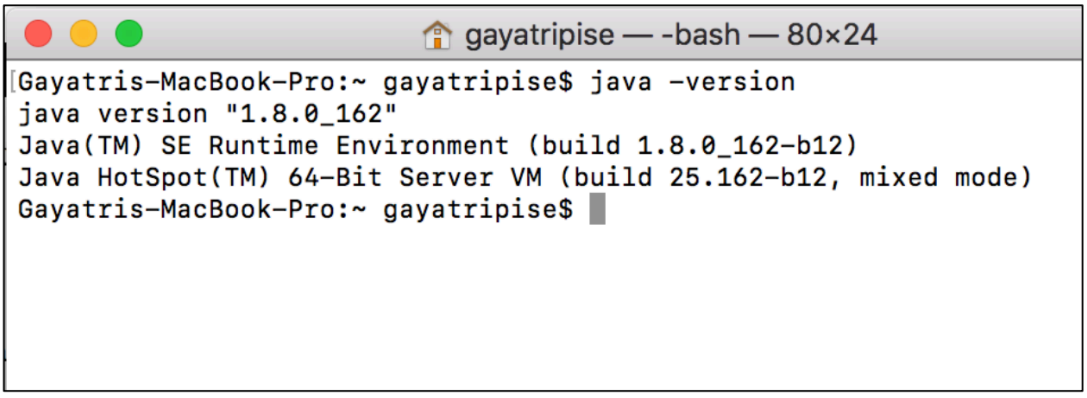


The Galactic Mail game is a single player game where an intergalactic mail carrier must deliver mails to a number of inhabited moons. The player is in charge of steering the mail carrier from moon to moon avoiding the moving asteroids in its way. Asteroids can move in any random direction and it damages the mail carrier on collision. The carrier can be steered using keyboard keys and the game is complete when the mail is delivered to all the moons.



2. Execution and Development Environment

- Java JDK 1.8
- Intellij IDEA
- Eclipse IDE
- Sublime Text



3. Scope of Work

i. Classes re-used in both the games.

We identified a few components and functionality common to both the games and called them as core game components. Our core components form the following - game menu, collision detection, game frame, game clock, game world, game object, sprite object, moving objects, destroyable objects, game application, key event handler, audio and so on.

ii. Classes for Tank Game.

The components specific to the Tank Game include wall object, destructible wall, indestructible wall, tank world, tank weapon, tank bullet, tank object and so on. The tank game map, tank animation sprite, bullet animation sprite and walls are read from provided image resources.

iii. Classes for Galactic Mail game.

The components specific to the Galactic Mail game include planet object, asteroids, space ship, galactic world and so on. The images and animation of planet, spaceship, asteroid and space map is loaded from image resource files.

4. Command Line instructions to Compile and Execute

i. Tank Game

```
javac src/tanks/Main.java
java src/tanks/Main
```

ii. Galactic Mail

```
javac src/galactic/Main.java
java src/galactic/Main
```

5. Assumptions

We made a few assumptions in both the games as follows:

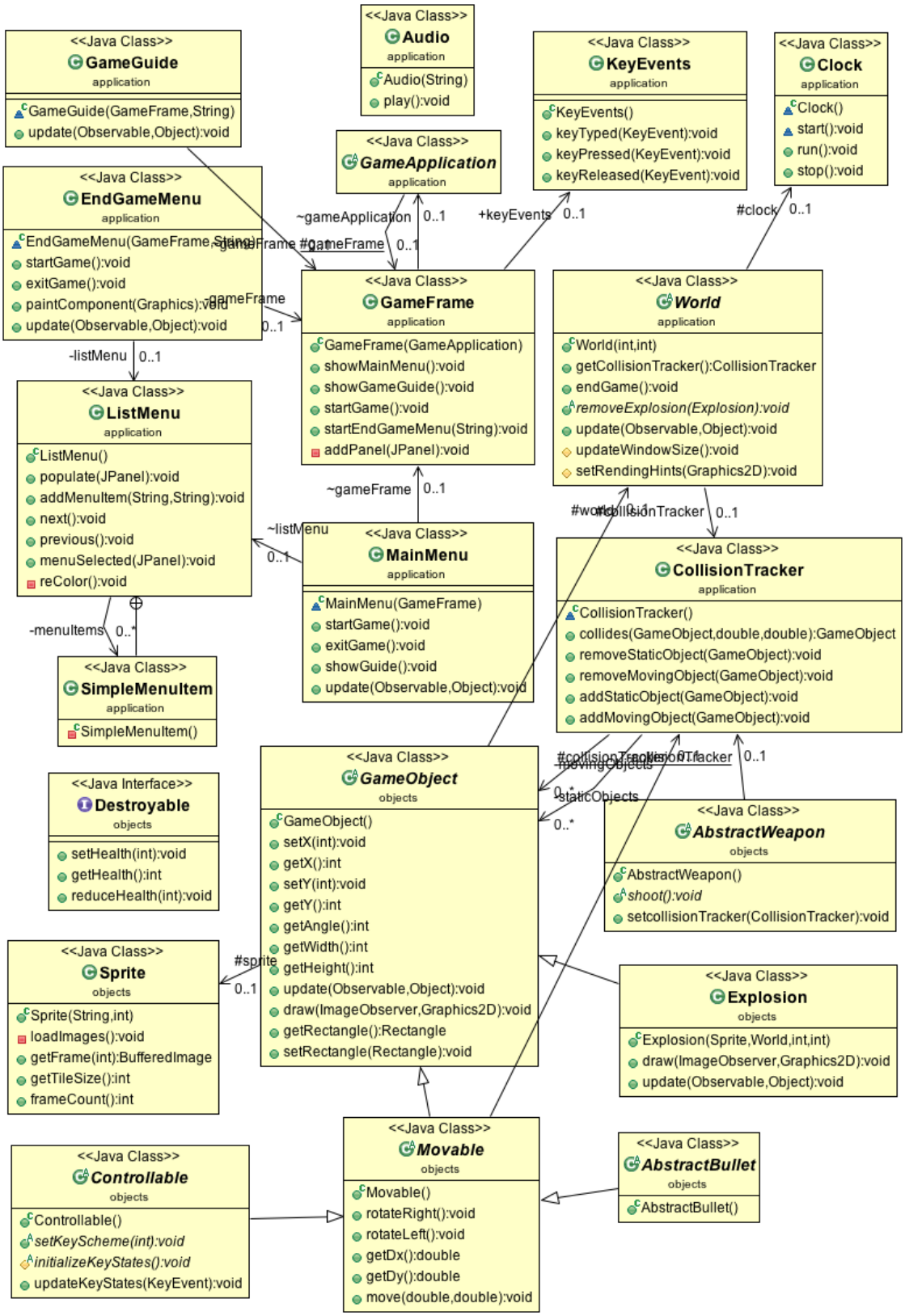
Tank Game:

i. The two tanks have a pre-determined life and every 5 bullets decrease the life by 20%.
ii. The game is terminated when any tank life reaches 0.
iii. The game always starts with the default layout and tank positions.
iv. Destructible walls are removed from the game after 3 collisions with the bullet.

Galactic Mail:

i. The space ship delivers the mail on visiting the moon.
ii. Asteroid do not harm the spaceship when spaceship is on a planet.
iii. The game is over when an asteroid hit the spaceship, or all the mails are delivered to the moons.

6. Implementation Details and Class Diagrams



The above diagram lists the classes used in the Tank game and Galactic Mail. These classes form the core of both the games and can be extended to many other games as well. The responsibility of each of the core class is described below.

i. **MainMenu**
This is the first screen shown when anyone starts the game. It consists of game options available to the user. Currently the options available are: start game, exit game and show guide. It implements an Observer for the keyboard key events.

ii. **EndGameMenu**
This is the screen visible to user after any player wins the game. It consists of game options available to the user after one game finishes. It implements an Observer for the keyboard key events.

iii. **ListMenu**
ListMenu is the provider class for MainMenu and EndGameMenu. It takes input and gives processed output to MainMenu and EndGameMenu. SimpleMenuItem is the object for each option available in MainMenu and EndGameMenu.

iv. **GameGuide**
This is the screen which displays help content to the user. It implements an Observer for keyboard key events.

v. **KeyEvents**
This class is an Observer class for any key presses in the game and implements a KeyListener. It notifies all the observers of any keyPressed or keyReleased event.

vi. **Audio**
This class is responsible to play the explosion audio after any collision.

vii. **Clock**
Clock is the Observable timer for all components of the game. It implements Runnable and notifies all the observers after fixed delay.

viii. **GameApplication**
This is the application class of the game and holds the GameFrame.

ix. **GameFrame**
GameFrame is the game UI. It extends JFrame and creates the game UI of fixed width and height.

x. **World**
This class is the game play panel and extends JPanel. It implements an Observable and redraws the game screen after every Clock event. It also maintains a CollisionTracker which tracks any object collision in the game.

xi. **CollisionTracker**
This class contains a list of all the static and movable objects in the game. The game objects are always updated with their current location and boundary. CollisionTracker tracks collision between any two objects present in the list.

xii. **Sprite**
Sprite object holds the array of all images required for an animation. And returns appropriate image depending on the animation requirement.

xiii. **Destroyable**
It is an interface for all the objects which can be removed from the game.

xiv. **Movable**
It is an abstract class which can be extended by any moving object in the game. Collision of any moving object is tracked by the Movable class.

xv. **GameObject**
This is the main game object and maintains the updated object position in the game. All the entities of game extend GameObject.

xvi. **Explosion**
This is the class which draws explosion on the screen. It holds the explosion animation Sprite and explosion co-ordinates.

xvii. **Controllable**
Controllable is the abstract class for any moving entity in the game that can be controlled by keyboard events. It extends the Movable object and processes any key events of the GameObject.

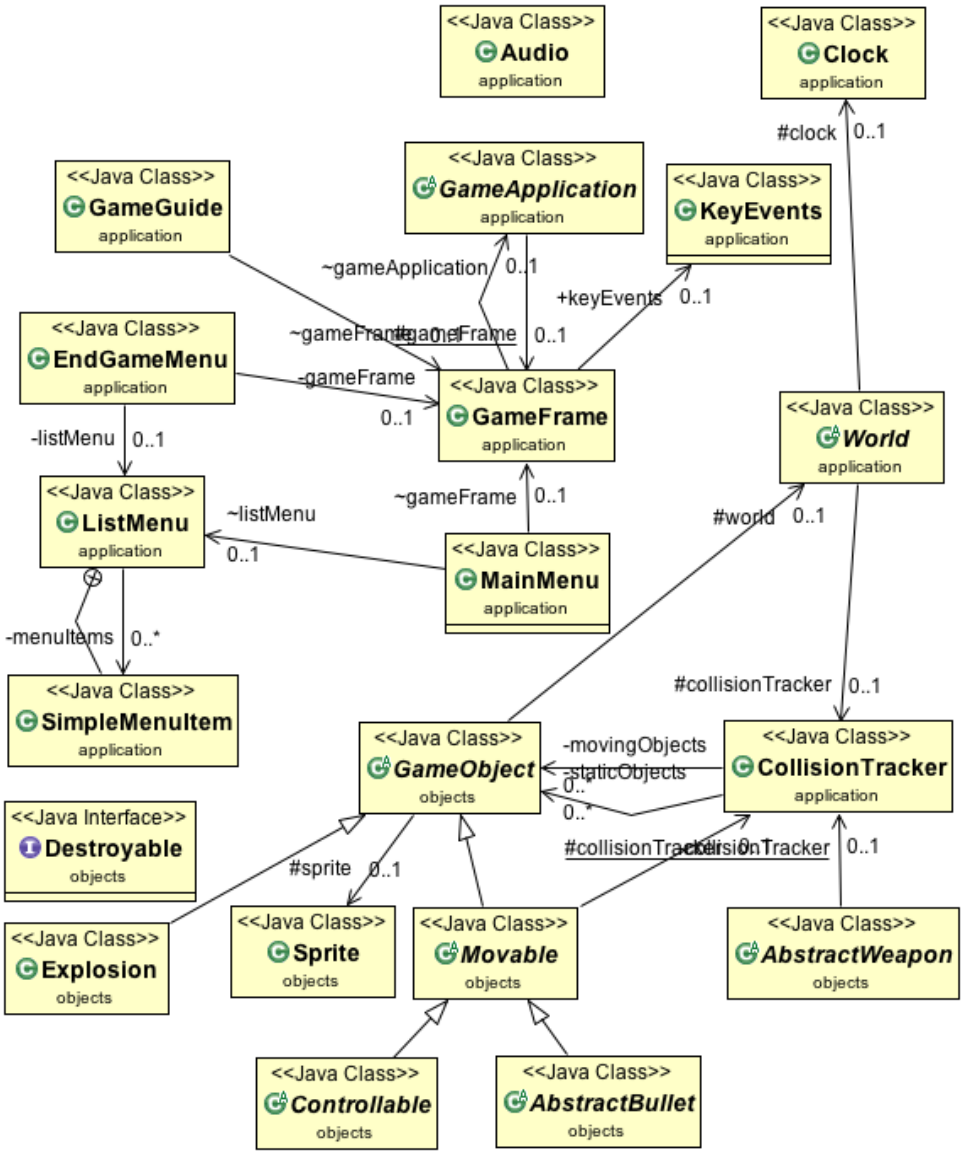
xviii. AbstractWeapon

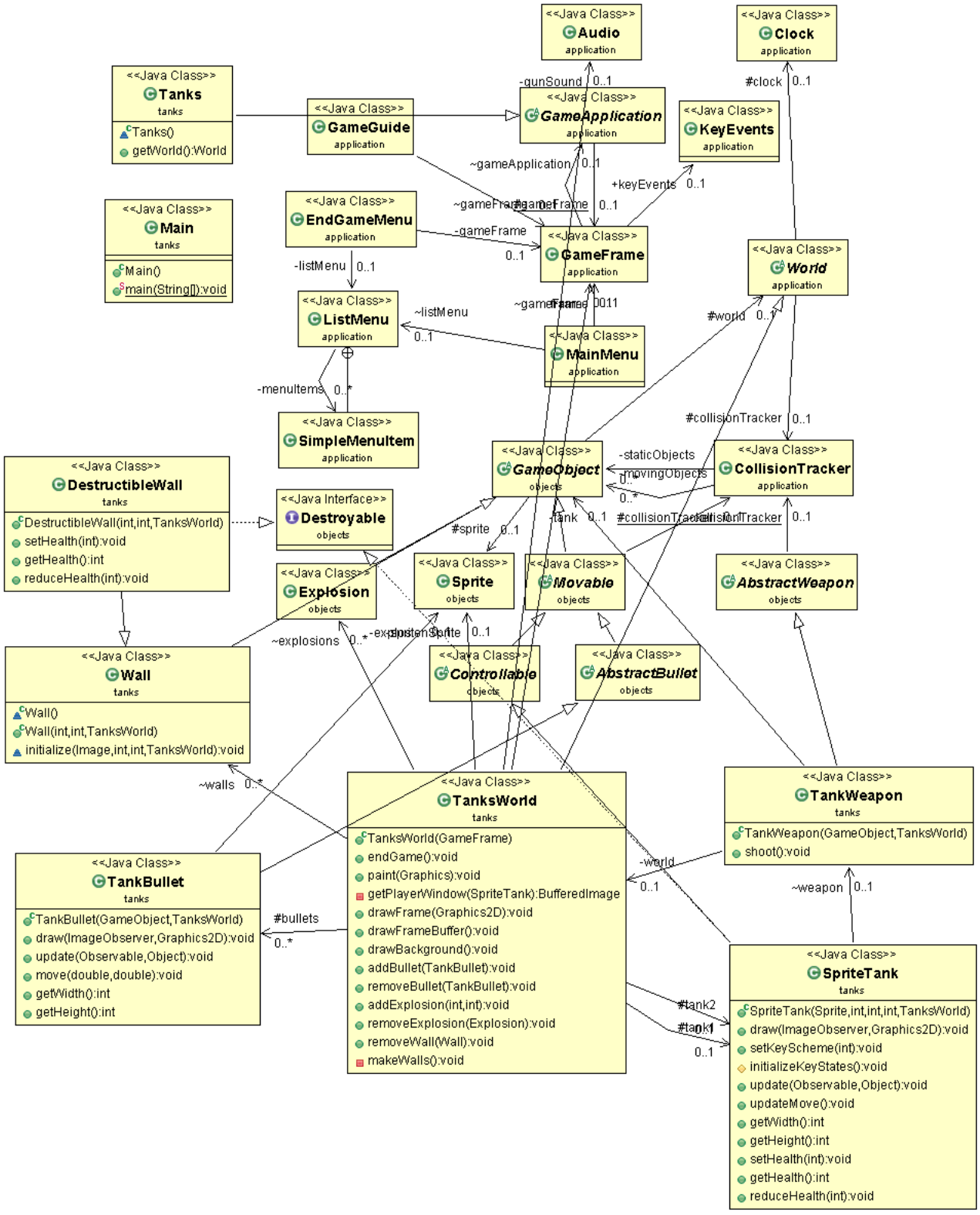
This is an abstract class available for any GameObject to provide the shoot functionality.

xix. AbstractBullet

This is an abstract class available for any moving Bullet object.

This is the short representation of the above described core game components and the shorter version is used in class diagrams of Tank game and Galactic Mail game.




<<Java Class>>
Audio
application
<<Java Class>>
Clock
application
<<Java Class>>
Tanks
tanks
Tanks()
getWorld():World
<<Java Class>>
GameGuide
application
-gunSound 0..1
<<Java Class>>
GameApplication
application
#clock 0..1
<<Java Class>>
KeyEvents
application
~gameApplication 0..1
+keyEvents 0..1
<<Java Class>>
Main
tanks
Main()
main(String[]):void
<<Java Class>>
EndGameMenu
application
~gameFrame 0..1
#gameFrame 0..1
-gameFrame
0..1
<<Java Class>>
GameFrame
application
-listMenu 0..1
<<Java Class>>
World
application
<<Java Class>>
ListMenu
application
~listMenu
0..1
~gameFrame 0..1
#world 0..1
<<Java Class>>
MainMenu
application
-menuItems 0..*
<<Java Class>>
SimpleMenuItem
application
#collisionTracker 0..1
<<Java Class>>
DestructibleWall
tanks
DestructibleWall(int,int,TanksWorld)
setHealth(int):void
getHealth():int
reduceHealth(int):void
<<Java Interface>>
Destroyable
objects
<<Java Class>>
GameObject
objects
-staticObjects
-movingObjects
0..*
<<Java Class>>
CollisionTracker
application
#collisionTracker
0..1
#sprite 0..1
-tank 0..1
<<Java Class>>
Explosion
objects
<<Java Class>>
Sprite
objects
<<Java Class>>
Movable
objects
<<Java Class>>
AbstractWeapon
objects
~explosions 0..*
-explosionSprite 0..1
<<Java Class>>
Wall
tanks
Wall()
Wall(int,int,TanksWorld)
initialize(Image,int,int,TanksWorld):void
<<Java Class>>
Controllable
objects
<<Java Class>>
AbstractBullet
objects
~walls 0..*
<<Java Class>>
TanksWorld
tanks
TanksWorld(GameFrame)
endGame():void
paint(Graphics):void
getPlayerWindow(SpriteTank):BufferedImage
drawFrame(Graphics2D):void
drawFrameBuffer():void
drawBackground():void
addBullet(TankBullet):void
removeBullet(TankBullet):void
addExplosion(int,int):void
removeExplosion(Explosion):void
removeWall(Wall):void
makeWalls():void
<<Java Class>>
TankWeapon
tanks
TankWeapon(GameObject,TanksWorld)
shoot():void
-world 0..1
~weapon 0..1
<<Java Class>>
TankBullet
tanks
TankBullet(GameObject,TanksWorld)
draw(ImageObserver,Graphics2D):void
update(Observable,Object):void
move(double,double):void
getWidth():int
getHeight():int
#bullets 0..*
#tank2
#tank1
0..1
<<Java Class>>
SpriteTank
tanks
SpriteTank(Sprite,int,int,int,TanksWorld)
draw(ImageObserver,Graphics2D):void
setKeyScheme(int):void
initializeKeyStates():void
update(Observable,Object):void
updateMove():void
getWidth():int
getHeight():int
setHealth(int):void
getHealth():int
reduceHealth(int):void

The Tank game utilizes all of the above core components and adds functionality specific to the tank game. Responsibility of each class in the Tank game is described below.

i. **Tanks**
This is the entry point for tank game and extends GameApplication. It starts the TankWorld which is the tank game play JPanel.

ii. **TanksWorld**
TanksWorld is the tank game play panel of the type JPanel. It extends World from the core components and adds the tank game UI. It redraws the updated game play panel on every Clock event. For the tank game, it initializes two SpriteTank objects.

iii. **SpriteTank**
This is the moving tank object in the game. It extends Movable and implements Destroyable from the core components. It initializes and renders the tank animation Sprite. SpriteTank also consists of a TankWeapon to shoot TankBullets.

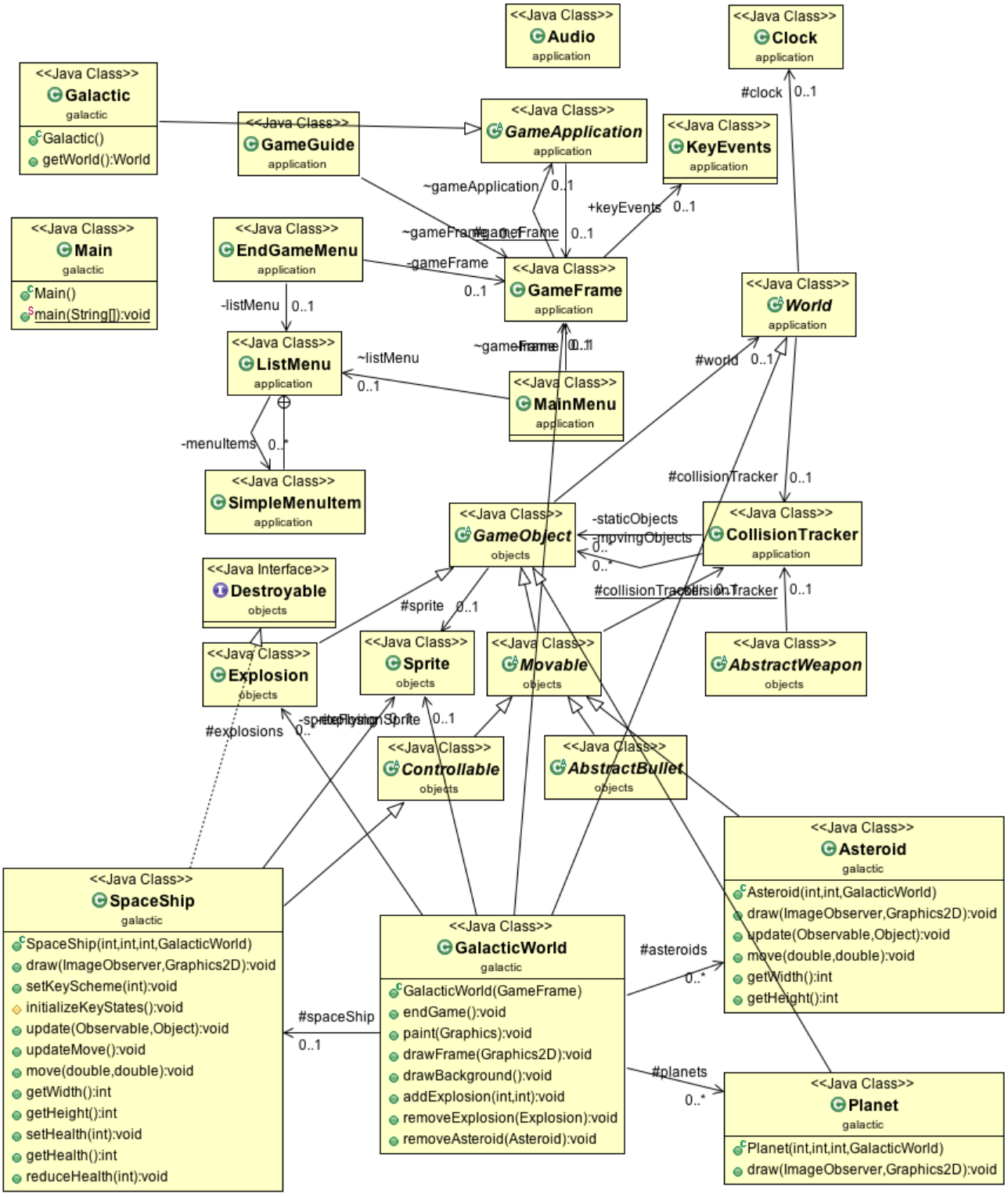
iv. **TankBullet**
TankBullet extends AbstractBullet and is the animated bullet that can be fired from the SpriteTank. TankBullet is also of the type GameObject.

v. **TankWeapon**
It is the weapon attached to the SpriteTank capable of shooting TankBullets. TankWeapon is of the type GameObject.

vi. **Wall**
This is the indestructible wall object in the TankGame. It is of the type GameObject and holds the current wall position in the game play panel.

vii. **DestructibleWall**
DestructibleWall is of the type Wall which implements Destroyable and can be removed from the game if its health reaches zero.

viii. **Main**
It is the entry point for the tank game when executing from command line.



The Galactic Mail game utilizes all of the core game components and adds functionality specific to the Galactic Mail game. Responsibility of each class in the Galactic Mail game is described below.

i. **Galactic**
This is the entry point for galactic mail game and extends GameApplication. It starts the GalacticWorld which is the galactic mail game play JPanel.

ii. **GalacticWorld**
GalacticWorld is the galactic mail game play panel of the type JPanel. It extends World from the core components and adds the galactic mail game UI. It redraws the updated game play panel on every Clock event. For the galactic mail game, it initializes Asteroid, SpaceShip and Planet objects.

iii. **Asteroid**
This is the moving asteroid object in the game. It extends Movable from the core components. It initializes and renders the asteroid animation Sprite.

iv. **Planet**
This is the planet object in the game. It extends GameObject from the core component. It initializes and renders the planet animation Sprite.

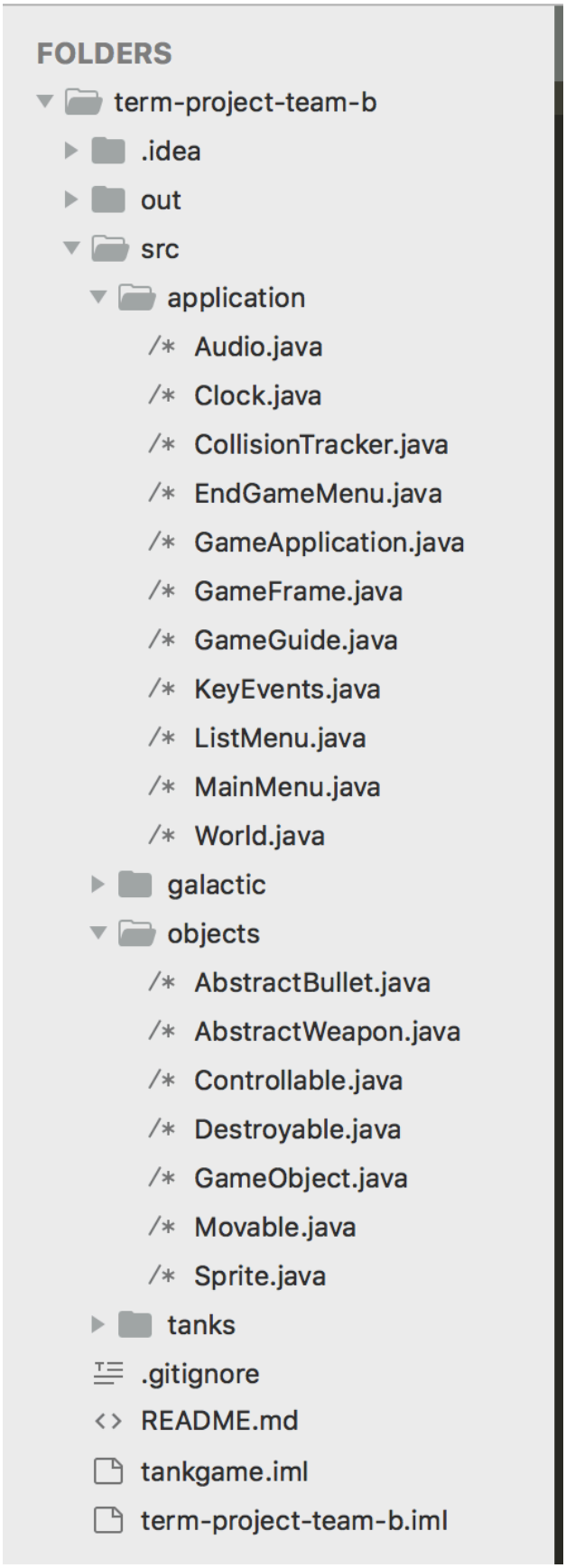
v. **SpaceShip**
This is the space ship object in the game. It extends Controllable and implements Destroyable from the core components. It initializes and renders the space ship animation Sprite.

vi. **Main**
It is the entry point for the galactic mail game when executing from command line.

7. Code Organization

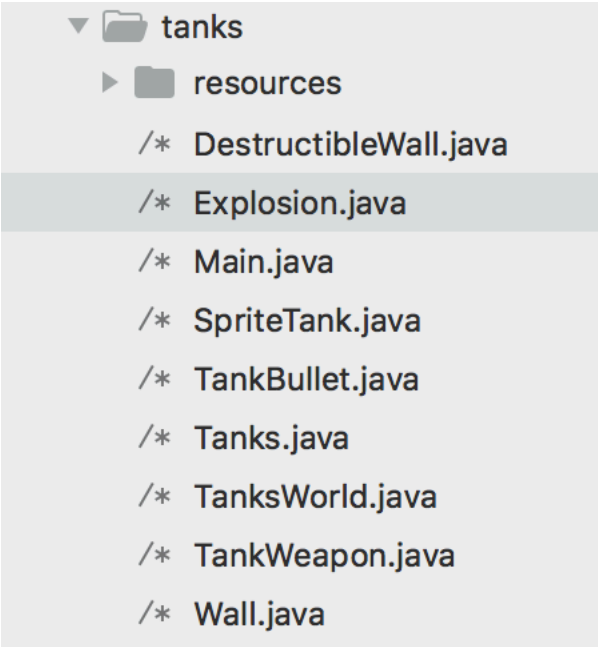
Our code is organized in the below format. The application package contains all the framework classes. These setups the game and use Java GUI and graphics components. The classes related to game menus are also included in the application package.

The objects package consists of all the basic game objects which could be extended by respective games and altered as needed.


FOLDERS
term-project-team-b
.idea
out
src
application
Audio.java
Clock.java
CollisionTracker.java
EndGameMenu.java
GameApplication.java
GameFrame.java
GameGuide.java
KeyEvents.java
ListMenu.java
MainMenu.java
World.java
galactic
objects
AbstractBullet.java
AbstractWeapon.java
Controllable.java
Destroyable.java
GameObject.java
Movable.java
Sprite.java
tanks
.gitignore
README.md
tankgame.iml
term-project-team-b.iml

Classes for Galactic Mail are in the galactic package.

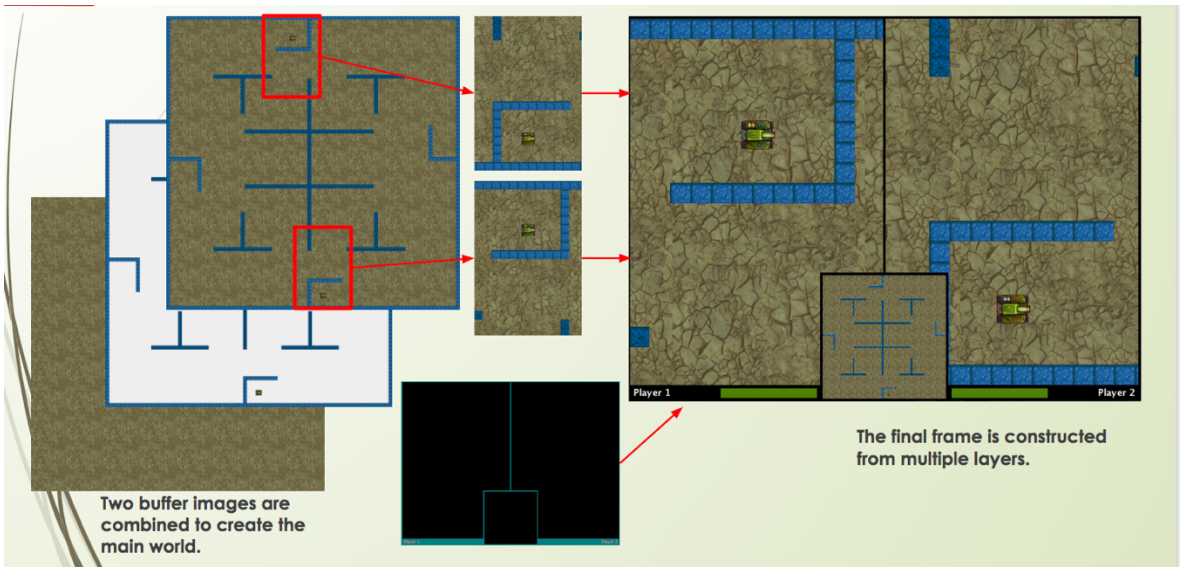
Classes specific to Tank Game are in the tanks package.



8. Conclusion

- By using object-oriented concepts, we could design a solution for the Tank Game and Galactic Mail game in such a way that major components could be re-used in both the games.
- Designing a collision tracking mechanism which could work for all game scenarios was an interesting challenge. We could achieve this by adding a CollisionTracker class which keeps reference of all moving and static objects and has a method to calculate intersection percentage of any two objects.

- The game map for multiplayer game is constructed by rendering multiple layers of images on a Graphics2D frame. And the map for each player is updated on every clock tick.



- To provide any update in the game, we used the Observer design pattern. All the game objects are observers to the Clock events.
- Being a group project, getting acquainted with the collaboration features of GIT was a good learning experience.
- Robust planning before implementation and utilizing object-oriented design patterns helped us design our solution.