Game 1: Tank Game

Project Information:

Ideally the purpose of this first game is to utilize all of the given requirements on iLearn and to make sure the code is reusable for the second game. The criteria includes, but is not limited to: performance, movement, mini-map, collision, a window for each player, an explosion effect, and life points.

Introduction:

The objective of this game is to destroy the other tank, but unfortunately at this time the game doesn't end. So if an individual would so please, they could play this game for all of eternity because theres check to end the game.

HOW TO START THE GAME: Ideally you want to be running the newest version of netbeans, if that happens to be the case all you need to do is press the play button.

HOW TO PLAY: Player 1 will use the wasd control scheme and to fire he will only need to tap space bar. While player 2 will use the arrow keys on the keyboard to move around the map and uses the numpad enter key to fire his tank. The goal of this game is to play for all of eternity, primarily because we didn't add a trigger to end the game.

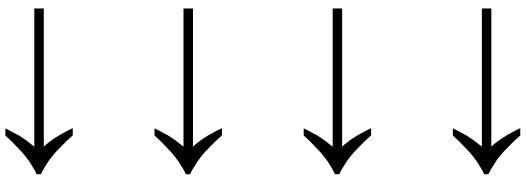
Assumptions:

At first glance, my team's approach to this game was to utilize the resources given to us in iLearn and to base our code off of what was given to us. After several days of planning we had created our initial UML diagram and we began coding. Our approach to this first project entailed first getting a map to be generated, then to program movement, and once those two functionalities were incorporated we were to implement features to our moving blobs on the screen. We assumed that by dividing up our work we would be able to finish this project in record time… Unfortunately, that's not what happened exactly.

Conclusion:

In the end, the tank game was completed but it was far from what it should have been developed into with the time we were given. Issues stemming from the lack of using github, conflicting code, and an inability to level with one another on certain aspects of this game plagued the development from day one. Fortunately, we got better at communicating and synchronized our coding sessions to allow us to keep track of where each one of us was taking the code (helped to avoid conflicts as well). But, the only thing we never used properly was github and it made our lives a lot more complicated and stressful than it probably needed to be.

The following pages contain a breakdown to the program's java members, functions, variables, and how they apply to each component of the game.



Tank_Audio.java

Summary:

As the name implies, this was the java class that handled all the music in our game. The basic code to write these functions was derived from: https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/package-summary.html

Data Fields:

Public:

Sounds: A public static final object of Tank_Audio.

Methods:

playLoop: Continuously play music on a loop over and over until the game exits.

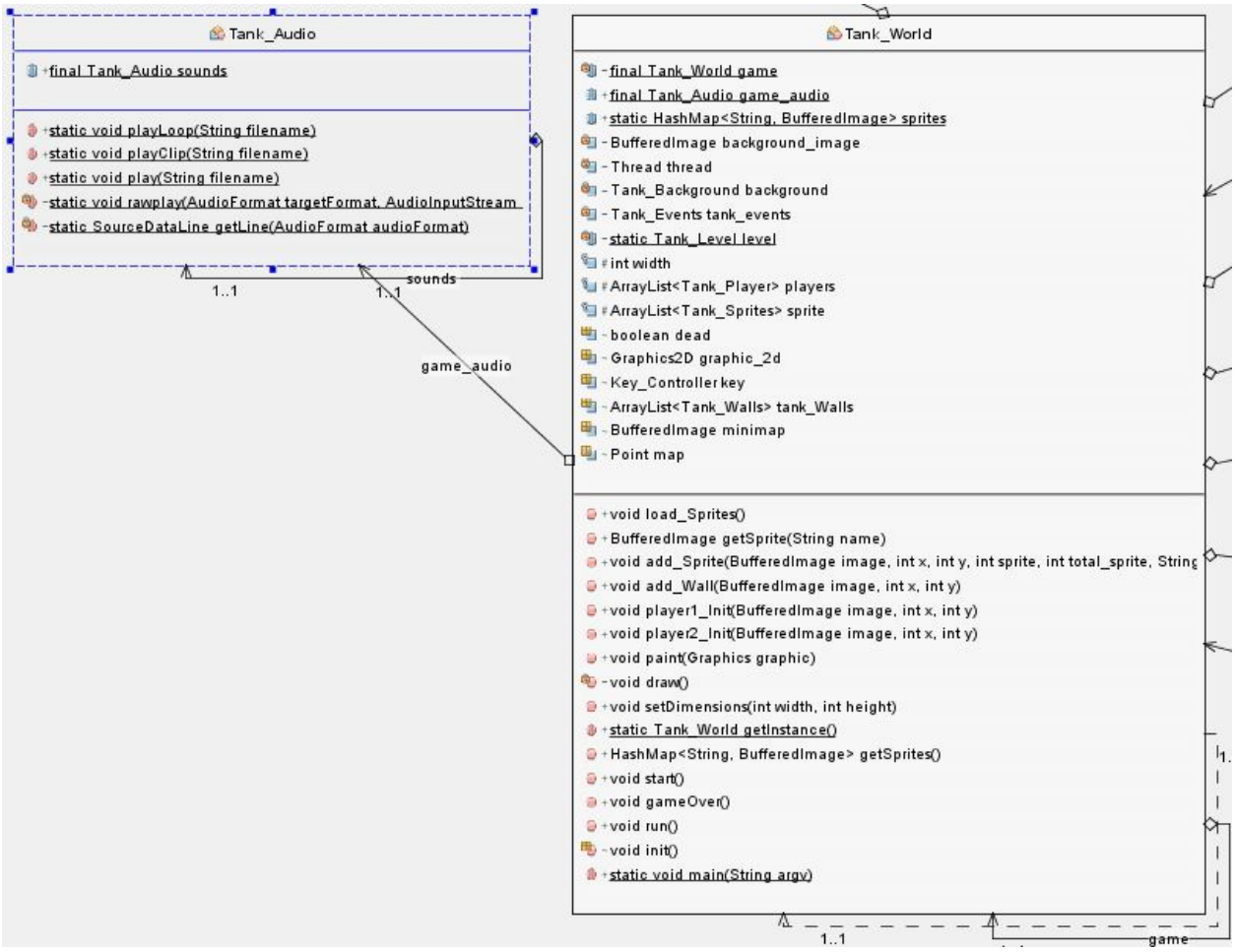
playClip: Play one short clip of music, in other words death sounds, bullet explosions, etc

Play: Creates a new thread so audio sounds can run simultaneously with other audio files, grabs audio file and decodes it, and when done the function starts.

Rawplay: AudioInputStream, obtain an audio input stream, write to external file, and convert to a different format.

SourceDataLine getLine: Writes audio bytes to a source data line, which handles the buffering and then delivers them to the mixer..

Tank_Audio Flow



Tank_Background.java

Summary:

This java class contains how background dimensions are measured, initialized, and drawn.

Data Fields:

Private:

Object:

Image_bg: An object of a Bufferedimage.

Integer:

Width: Took the width of the image.

Height: Took the height of the image.

Methods:

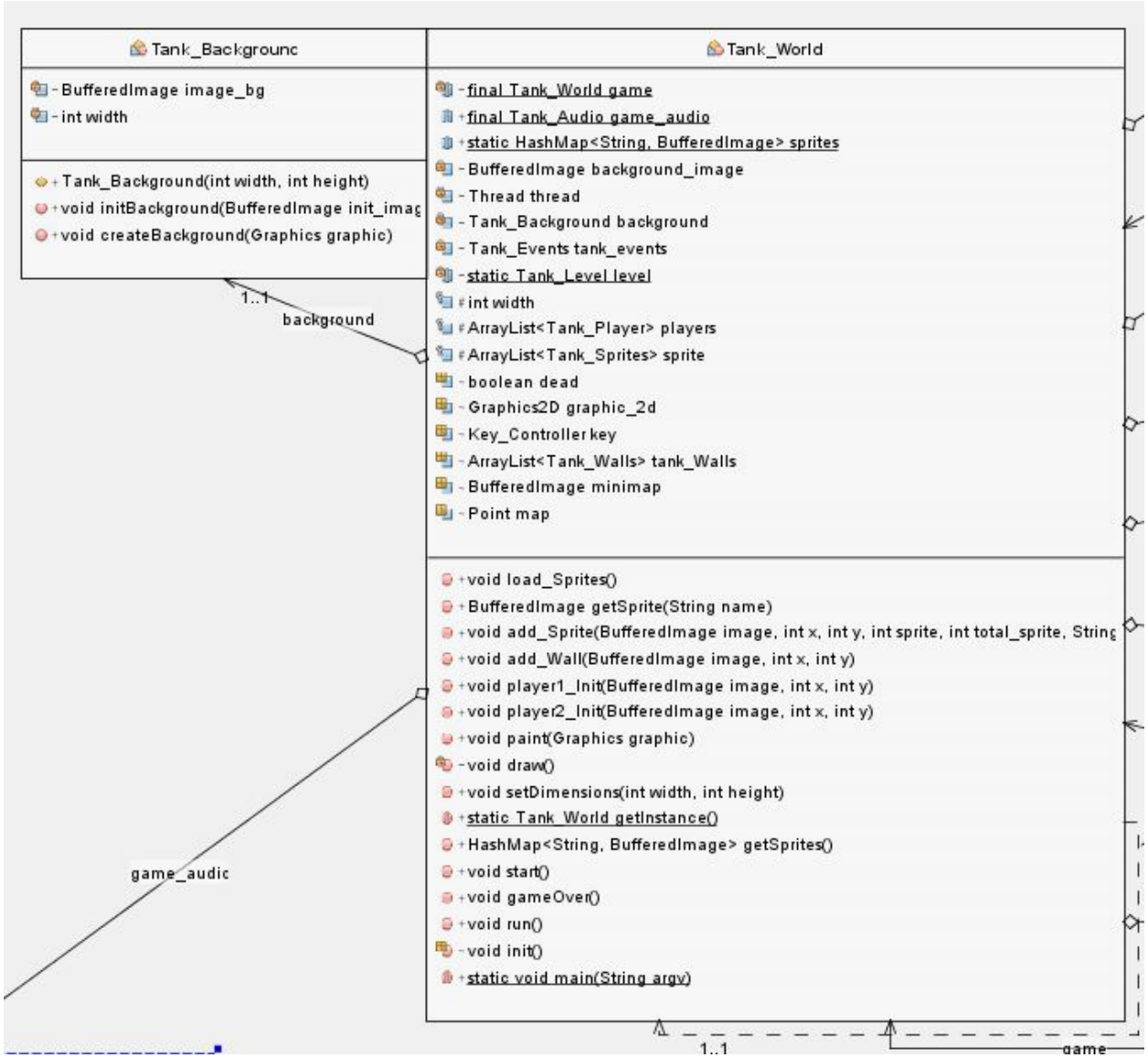
Public:

Tank_Background: Constructor, applies width and height.

initBackground: Takes a buffered image and passes it to image_bg.

createBackground: Draws the image based off of width, height, and image.

Tank_Background Flow



Tank_Background_Object.java

Summary:

Tank_Background_Object is a java class that utilizes a created background and draws it to the background.

Data fields:

Public:

Integer:

X: X position of image

Y: Y position of image

Object:

image= background image, buffered.

Boolean:

Visible: Test to see if constructor for object has been called and its visible...

Methods:

Tank_Background_Object: Takes image, x, and y as parameters to create a Constructor and initialize values.

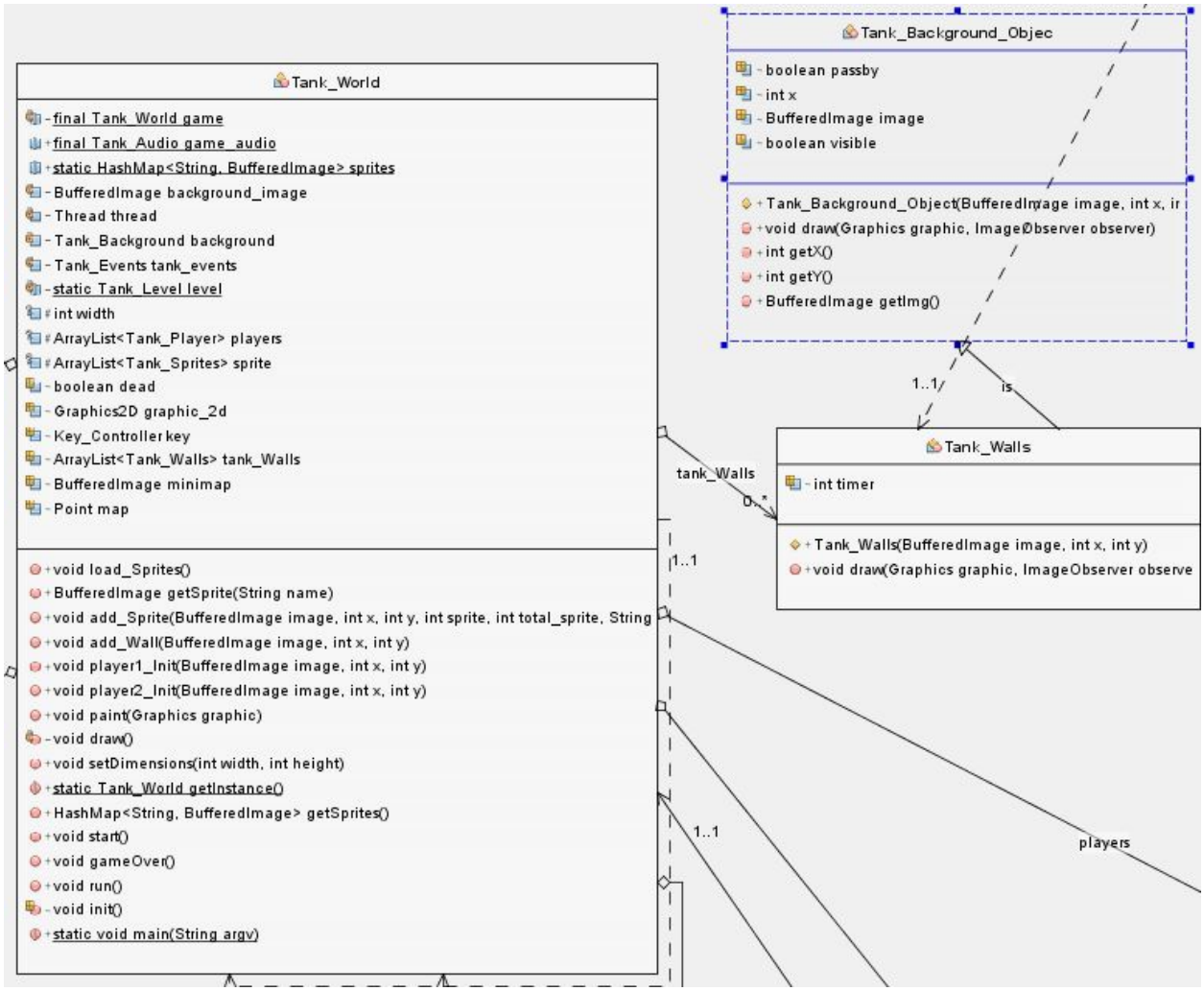
Draw: Takes graphic and observer as parameter and then draws an image based off of a constructor class.

Get:x Returns x.

Gety: Returns y.

getImg: Returns the image.

Tank_Background_Object Flow



Tank_Damage.java

Summary:

The purpose of the tank_damage class is to be able to shoot bullets, draw them on the image, and make sure they interact with other images on the map.

Data Fields:

Public:

Integer:

Movement: Traces movement of the bullet

X: The x position of the bullet

Y: The y position of the bullet

Object:

B: Background image, poorly named, that will take the sprite.

Boolean:

Visible: Test to see if bullet is still visible on the map.

Methods:

Tank_Damage: The constructor, takes image, x, y, and movement to apply to the sprite to be bullet.

Draw: Taking a graphic and observer parameter we use the values b (background), x, y, and observer to draw the image to the map.

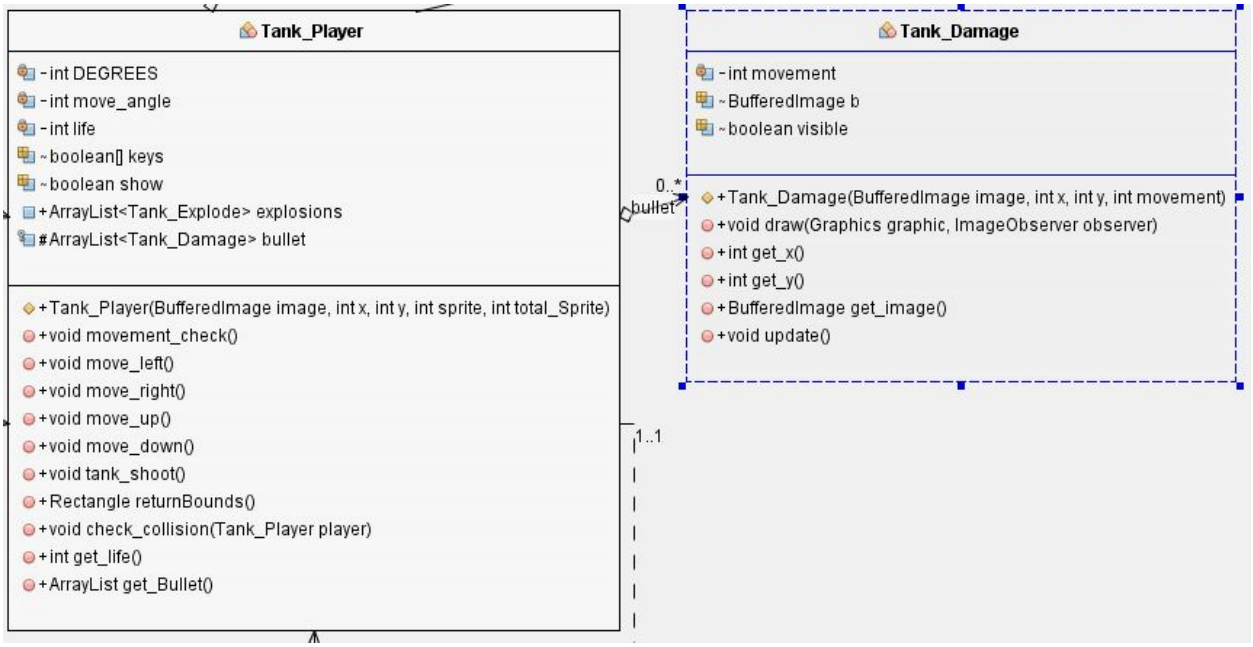
Get_x: Acquire the x value of the bullet

Get_y: Acquire the y value of the bullet

Get_image: Get the buffered image.

Update: Provided the bullet is moving, using the x and y values, we progress them along the map using Math.toRadians based off of their movement to shoot the bullets across the map.

Tank_Damage Flow



Tank_Events.java

Summary:

The purpose of tank events is to keep track of the movement of the tanks, their position on the map, and to check for collision between ALL the objects on the map.

Data Fields:

Private:

Integer[]:

Player_1: Stores both the x and y position of player one. In turn this allows for the tanks to be tracked on the map.

Player_2: Stores both the x and y position of player two.In turn this allows for the tanks to be tracked on the map.

Public:

Object:

Tank: Creates an instance of the tank world. This will allow for us to check to see if players are dead or not on the map. If so a function gameover will be called and the game will reset.

Methods:

Pressed: Checks to see if a key is pressed with the (WASD and ARROWS), if this is true it registered the key as “pressed.”

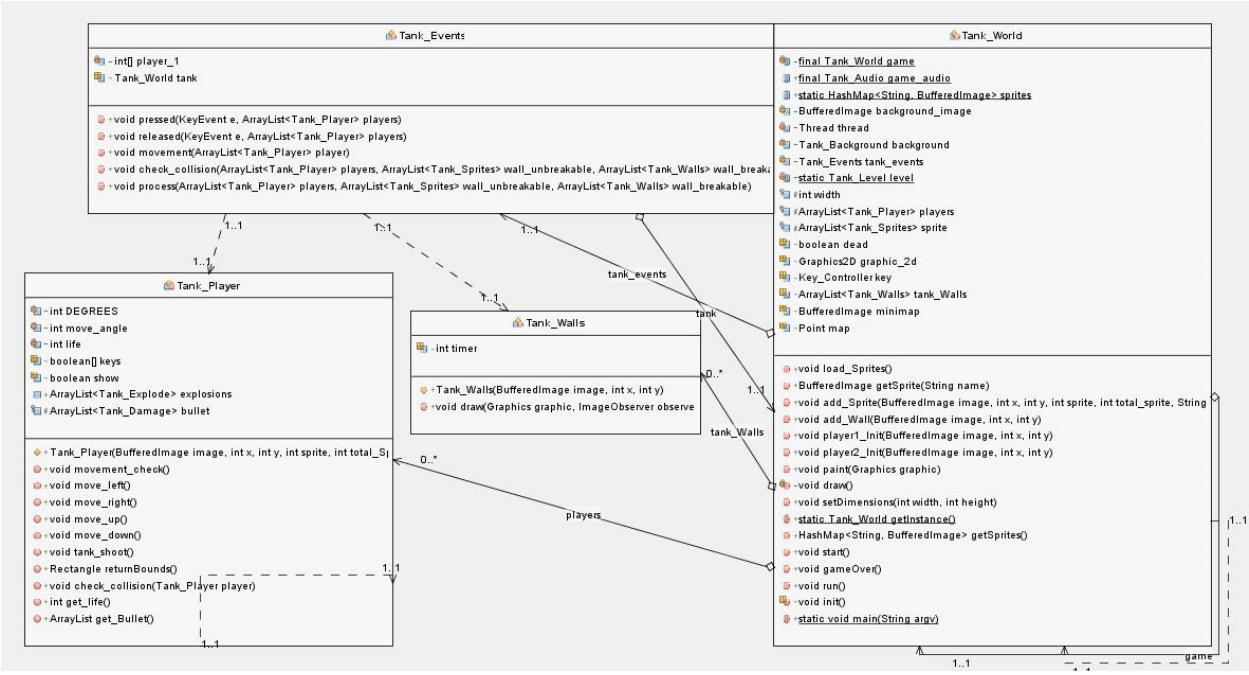
Released: Checks to see if any keys have been released since last checking. If this is true it registers the key as “released.”

Movement: Declares player_1 and player_2 as new integer arrays, this will allow this function to track the movement of the two players on the map. Once their locations have been updates a movement check will be incorporated to see which direction they’re moving.

Check_Collision: Using players and wall types, this function checks to see if their are any points on the map where the sprites of the tanks, walls, and bullets intersect with one another. If any of these checks become true, there will be an action in turn (damage, push the image away from the other, and etc).

Process: Taking the parameters of players and walls, checks for movement and collision among all images on the map.

Tank_Events Flow



Tank_Explode.java

Summary:

As the class name implies, we have a java class that is dedicated to drawing explosions for the tanks as well as the bullets upon impact.

Data Fields:

Public:

Integer:

//**Note the naming conventions are wrong here**

Vulnerable : Used to keep track of the number of explosions

Invulnerable: Used to make sure after three explosions vulnerable to rest and the tank's sprites are incremented.

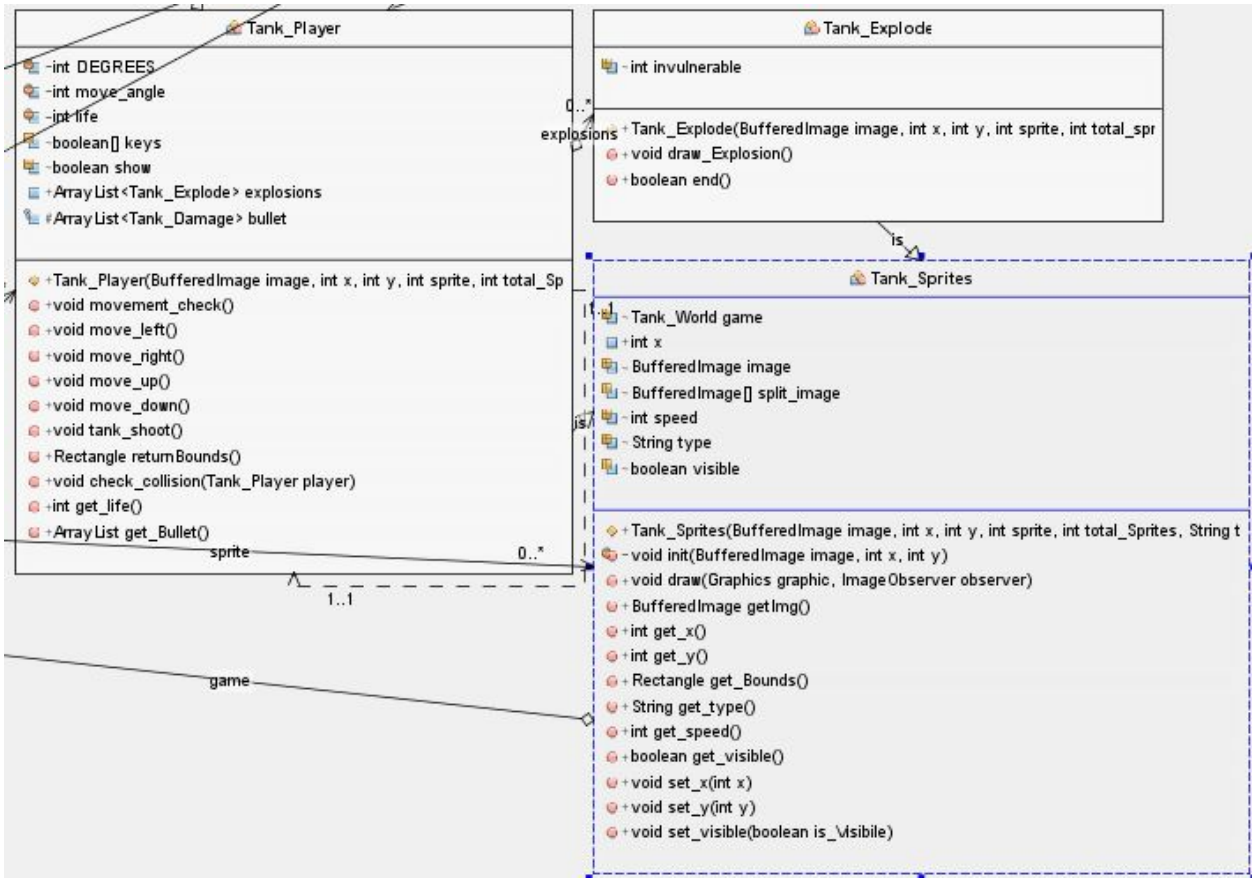
Methods:

Tank_Explode: Takes the bufferedimage, x, y, sprite, total sprites and assigns them to the superclass and assignes values for vulnerable and invulnerable.

draw_Explosion: Draws an explosion on the map, resets the tank after three explosions.

End: Only used once in the entire program, used to check for whether or not an explosion should be drawn. Simply a boolean method for true/false.

Tank_Explode Flow



Tank_Level.java

Summary:

Generates the level which the tanks will play on. Map generating is based off of the text file that is included in the tank resources folder and can be easily modified to generate any number of maps. Each piece of the map has a number associated with it to allow for easy modification of both this game and the next game to be created (pieces of the map, walls, tanks, koalas, etc).

Data Fields:

Public:

Object:

Level: The map to be generated which will be given from the file (text file).

String:

File: The file that is pulled from the resources to generate the map.

Int:

Width: The width of the text file for use in generating the walls, players, and power ups.

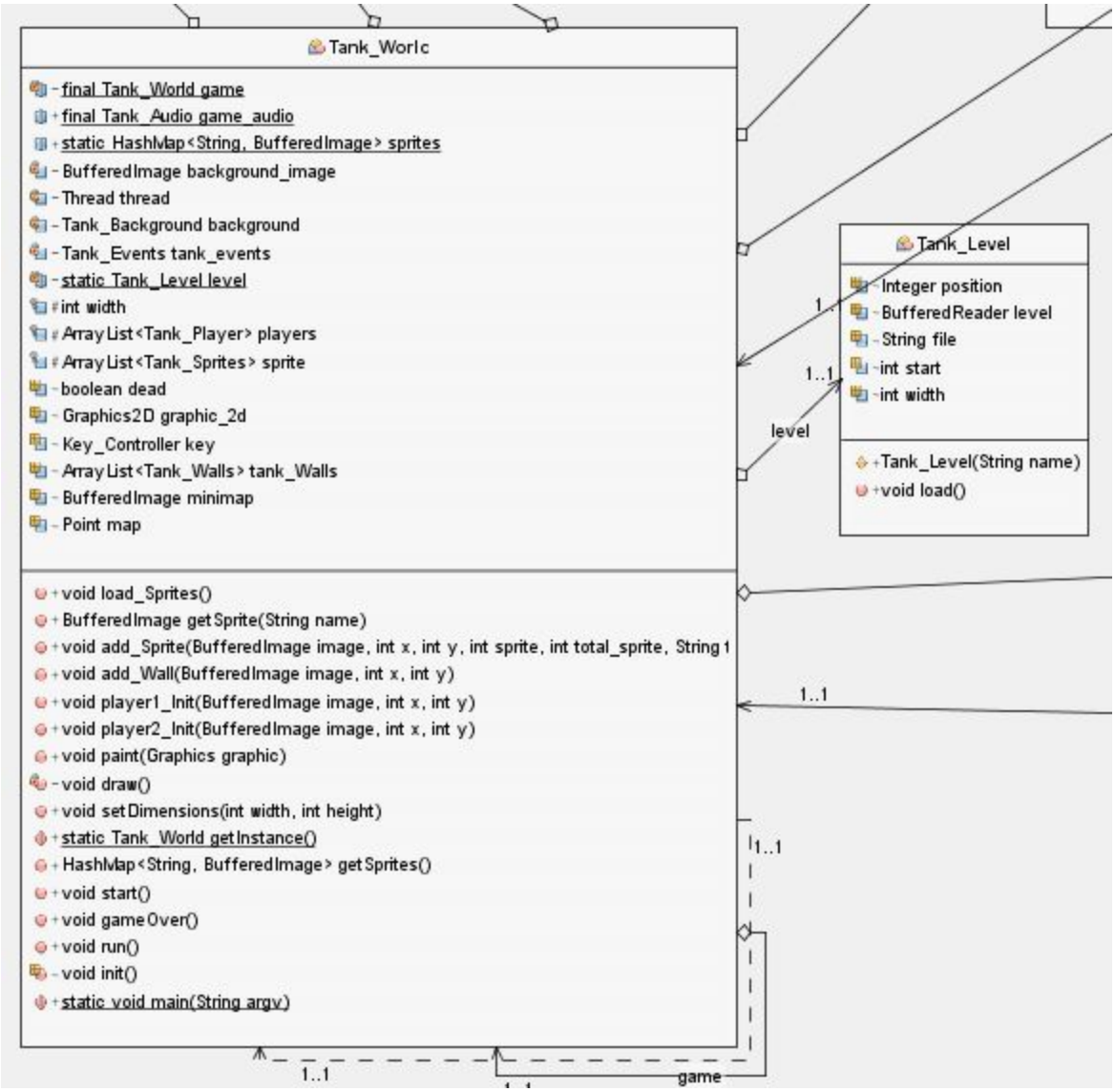
Height: Take the height of the text file for use in generating the walls, players, and power ups.

Methods:

Tank_Level: Constructor class, just needs a file name and then it can generate the map and assign it to the level object.

Load: Loads the map based off of the text file and utilizes a switch to associate each sprite to load onto the map. Once finished it closes level and the map has been loaded.

Tank_Level Flow



Tank_Player.java

Summary:

Gives the tank all of its functionality, in terms of movement, shooting, and collision.

Data Fields:

Int:

Degrees: Gives the tank its range of movement based off of radians

Move_angle: Based off of degrees we are able to obtain our tank's sprite movement.

Player_x: ***NOT USED, DEBUGGING PURPOSES ONLY.***

Player_y: ***NOT USED, DEBUGGING PURPOSES ONLY.***

Player_sprite: ***NOT USED, DEBUGGING PURPOSES ONLY.***

Life: Used to keep track of the tank's life throughout the game.

Points: Keeps track of how many kills you've obtained throughout your game.

Array Lists:

Explosions: Used to keep track of where to add explosions on tanks and walls.

Bullet: Used to keep track of where the bullets are and when they intersect with objects to remove them and to potentially cause damage.

Boolean:

Keys: Movements are recorded here, if a certain key is found to be true we call a move function based off of the key found to be true.

Show: Is the player visible, check.

Powerup: When intersecting with a powerup, if found to be true it will be removed from the map and a tank will be given a powerup.

Methods:

Tank_Player: Constructor for the player class takes the buffered image, x, y, sprite, and all sprites. Calls super and applies the given values, assigns all keys to start off as FALSE while allocating bullet, explosions, and move_angle with initialized values.

Movement_check: Based off of the keys being pressed, certain movement functions will be called for each tank (WASD for player_1 and ARROWS for player_2).

Move_left: If a sprite exceeds a certain angle the sprite will reset (avoids janky appearance of movement).

Move_right: If a sprite exceeds a certain angle the sprite will reset (avoids janky appearance of movement).

Move_down: Utilizes radians based off of the move angle to decipher how the sprite will react based off of SPEED.

Move_up: Utilizes radians based off of the move angle to decipher how the sprite will react based off of SPEED.

Tank_shoot: Adds a bullet to the map, if a powerup is available another bullet is added to the map. When a tank shoots you will see a slight offset due to the time required for the check between when the first bullet shoots and the second is cleared to fire.

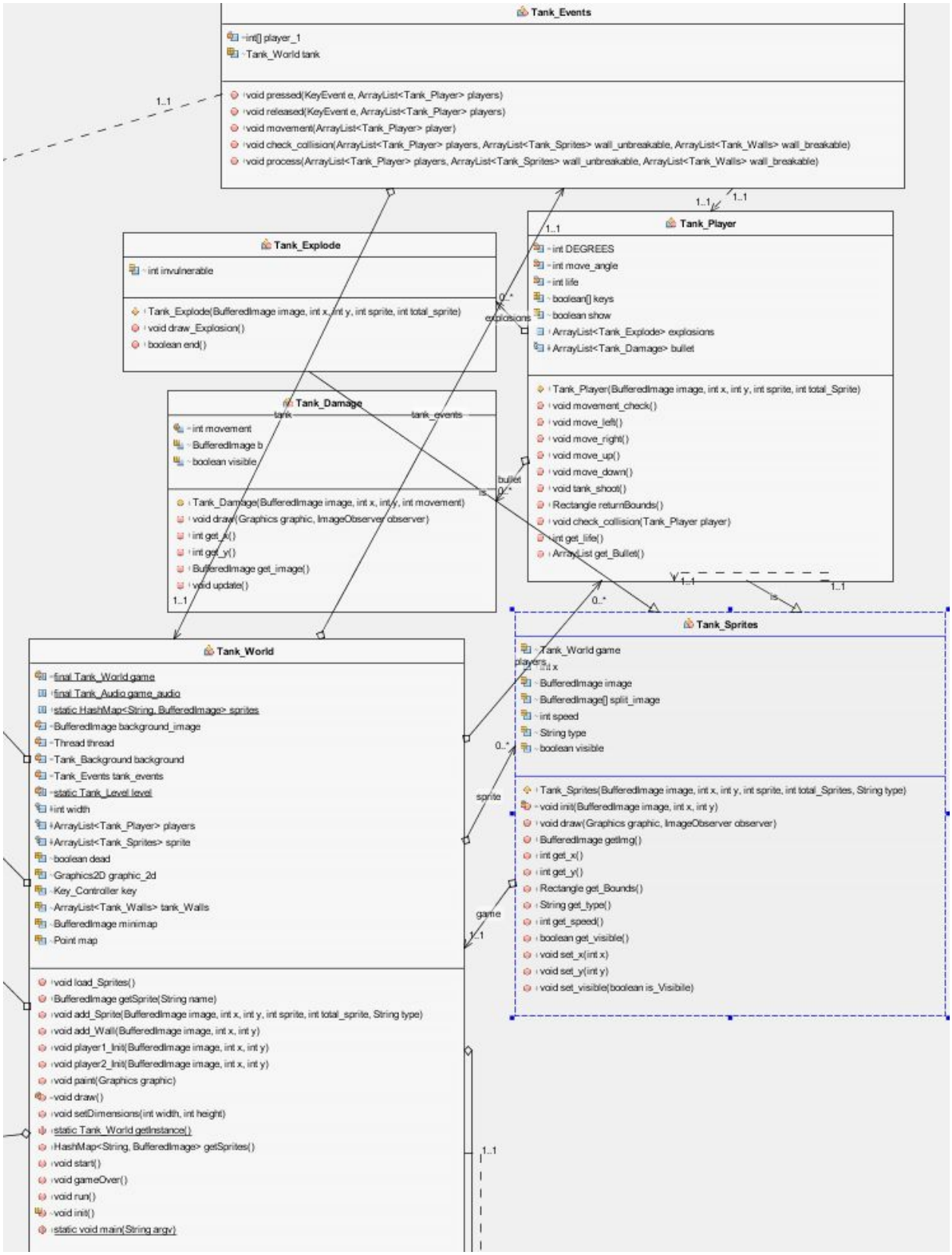
returnBounds: Gives the bounds of the current player.

Check_collision: Checks for collision between the tanks, walls, bullets, and power ups. When found to be true or false for each check the appropriate functions are called.

get_Life: Returns life.

get_Bullet: Returns bullet.

Tank_Player Flow



Tank_Sprites.java

Summary:

The tank sprites java file is used to initialize the sprites. It pools all the information from the sprites and allows for them to be easily accessible throughout the program. This also happens to be where the split screens and minimap are generated.

Data Fields:

Public:

Objects:

Game: Allows for an instance of the Tank_World to be created so that you can make references in tank_world.

Image: Generates the mini-map.

Split_Image: Splits the game image so you can have both a player_1 view and player_2 view.

Int:

Speed: Checks for the speed of sprite

String:

Type: Checks for type of sprite

Boolean:

Visible: Checks for visiblity of a sprite.

Methods:

Tank_Sprites: Constructor, takes an image, x,y, sprite, all sprites on map, and type. Assigns them accordingly to the class.

Init: Initializes the split screens.

Draw: Draws the tank split screens.

getImg: Gets image object.

Get_x: Gets x.

Get_y: Gets y.

get_Bounds: Gets the bounds of the sprite.

Get_type: Gets type.

Get_speed: Gets speed.

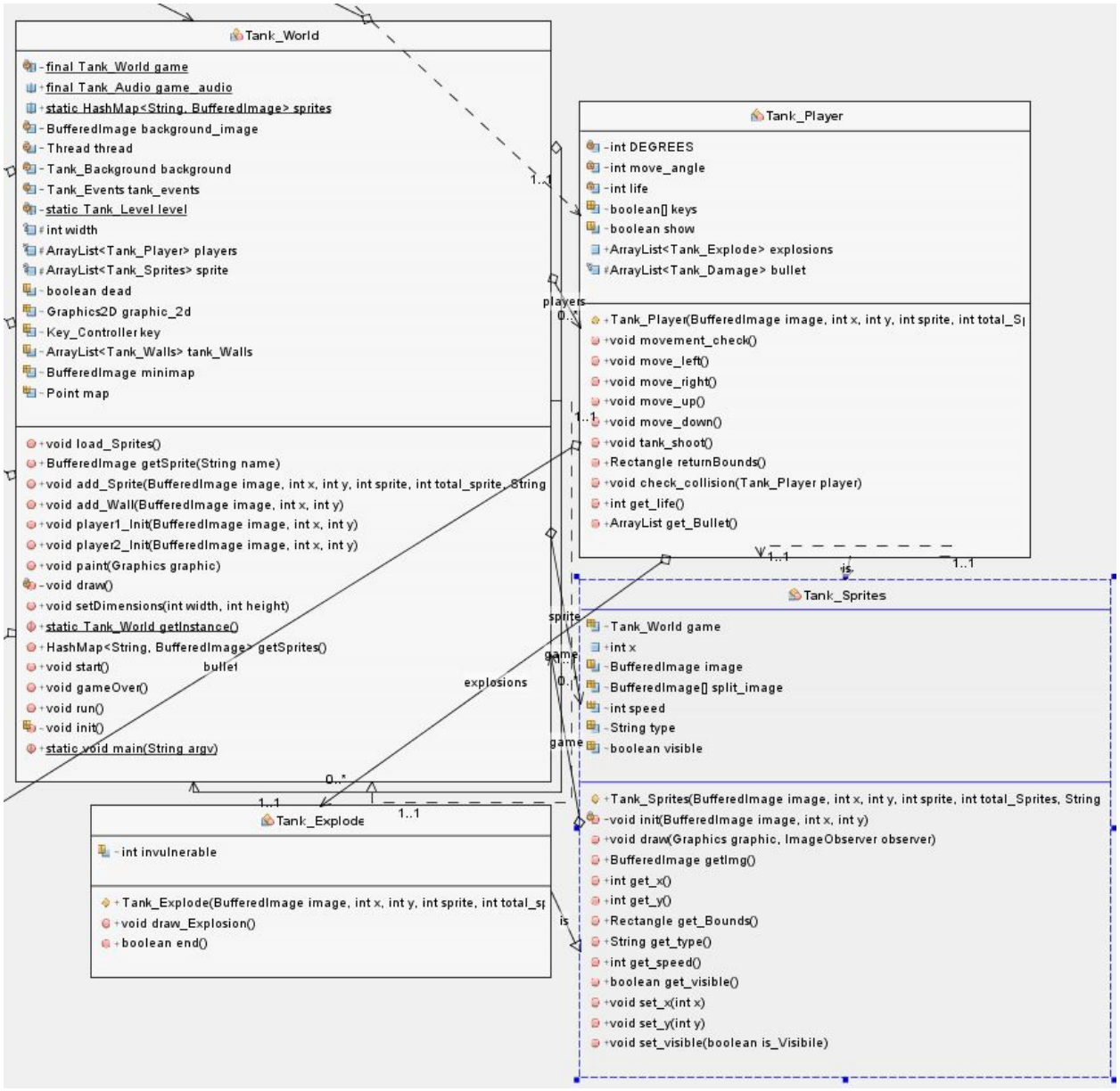
Get_visible: Gets visibility.

Set_x: Sets x, no erro checking.

Set_y: Sets y, no error checking.

Set_visible: Sets visibility, no error checking.

Tank_Sprites Flow



Tank_Walls.java

Summary:

Simply checks to see if walls exist, if not then we regenerate them after a short period of time.

Data Fields:

Public:

Int:

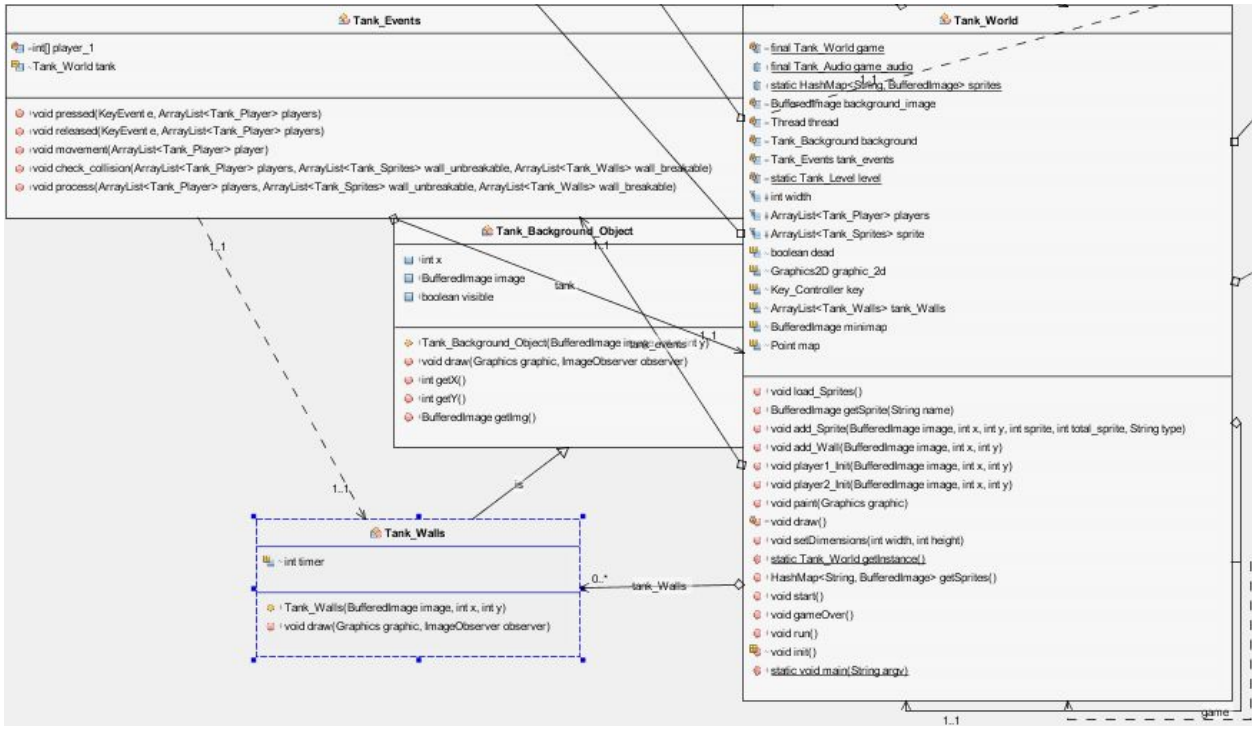
Timer: Tells you how long to wait in miliseconds until you regenerate a new wall.

Methods:

draw: Draws a new wall on the map if visiblity is false AND if the timer is less than zero

Tank_Walls: Takes image, x, and y and assigns them to the super class. Visiblity is by default marked as true.

Tank_Walls Flow



Tank_World.java

Summary:

An accumulation of everything in this program all linked by this java class. This is where all of the resources are allocated and distributed through out the fu

Data Fields:

Public:

Objects:

Game_audio: Creates an object of game audio which will allow it to be referenced throughout the program to play various audio files.

Grahpic_2d: Used to create the background image and to draw ALL the sprites (walls, players, and powerups).

Key: Used primarily to remove keys and to add new keys.

Minimap: Takes the background_image and then scales the instance based off of the image.

Map: Used to draw the bounds of the map and is also used to pass its variables to Tank_Background.

Hash Map:

Sprites: Hash map used to associate the names of the sprite with the image in the resource folder. After load_sprites is called from this point forward sprites will be used to manipulate sprites throughout the map.

Boolean

Dead: Tank is either dead or alive, simply a marker to restart the game.

ArrayList

tank_Walls: Used to add new walls, directly passed to tank_events.process to process the three array lists.

Private:

Objects:

Game: Used as an instance of Tank_World and is used to restart the game.

Background_image: Used to draw both the player1's view and player2's view onto the map.

Player1_view: Takes background image and is applied to player1.

Player2_view: Takes background image and is applied to player2.

Background: Used primarily in the init function which calls Tank_Background and utilizes map's values to help generate the background.

Tank_events: Used to process all events in the game such as keypresses and processing all players/sprites/walls.

Level: Takes the map resource given and is applied to the level object to be manipulated throughout the program.

Thread:

Thread: Used to run mulitple threads simulatenously, primarily used to repaint the screen while the tanks run amuck.

Protected:

Int

Width: Map width.

Height: Map height.

Player1_score: Player 1's score.

Player2_score: Player 2's score.

Array List:

Players: Used to keep track of player's location on the map and is passed as an observable object to check to see when keys are pressed to move the tanks.

Methods:

load_Sprites: Loads all of the sprites.

getSprite: Gets sprite.

Key_Controller: The call to check to see if a player is moving or not moving (for both players).

add_Sprite: Add a sprite to the list of sprites.

add_Wall: Add a wall to tank_walls.

Player1_init: Initalizes player1's values.

Player2_init: Initalizes player2's values.

Paint: Paints on the minimap, split screens, and score board.

Draw: Since this is the super class, everything is drawn here in this function.

setDimensions: Sets the dimensions of the map (width and height).

getInstance: Gets an instance of the tank_world.

getSprites: Gets the game's sprites.

Start: Starts the multi-threading process.

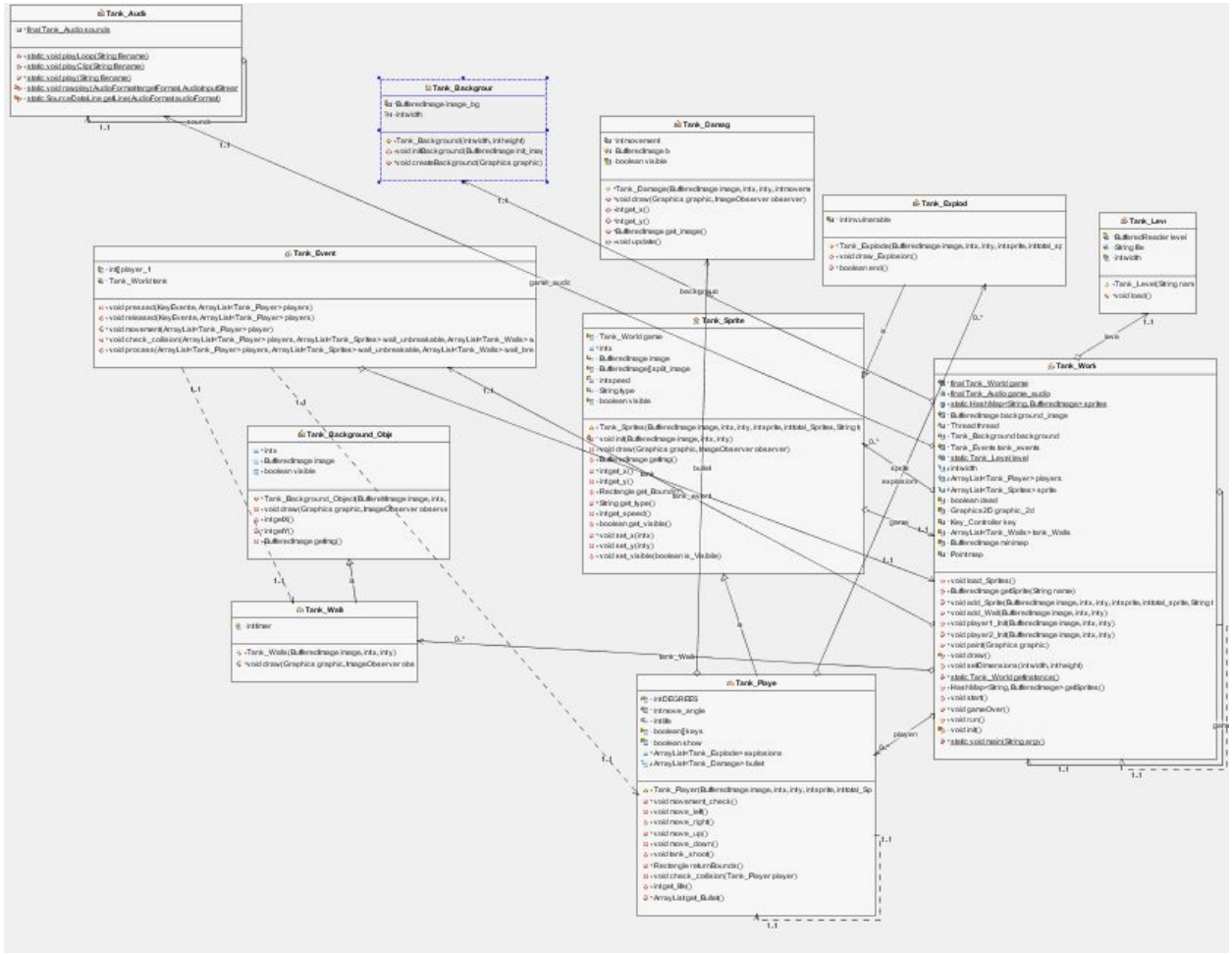
gameOver: When a tank has died, the game over function is called and everything on the map including the tanks are reset.

Run: Allows for the current thread to repaint every 25 ms.

Init: Sets the focus of the window on the jframe just opened by main. Begins the process of creating the sprites and map. Once this is completed the background is generated along with a keylistener to begin playing the game.

Main: Creates the jframe we used to play the game on, sets the game audio, dimensions, and begins the game.

Tank_World Flow



Game 2: Koala Escape

Project Information:

In the second game of the project, we have been directed to create a new game based off of the previous games' code. What had to be done was the making of a new game with smooth performance and pleasant user experience. Like the previous game, we wanted a game that would be reusable and that would have a clean design. Now that we had a base knowledge on creating these 2D games and had a lot of the code fresh and ready, creating the Koala game was a lot of tweaking of the tank game, but there were also new methods and mechanisms we had to explore and implement.

Introduction:

The object of the game is to save all of the koalas. You have to get them to the exit without any of them being exploded by the tnt. The tricky part about the game is that all the koalas move at the same time, meaning you can not control them one at a time. This, combined with the map being a maze provides for a brain busting thriller of a game.

HOW TO START THE GAME: To start the game, clone the repository and create a new project in NetBeans using the code from the repository. Build and run the project and the java application will appear.

HOW TO PLAY: Begin by clicking the "Start" button. Use the direction keys on your keyboard to move the koalas. You want to avoid running into the tnt with the koalas or you will not be able to move onto the next level. In the case that you do lose a koala to the tnt, you must click the restart button to restart the game. In the case that you do get all of the koalas to the exit, you will move on to the next map, and if you get them all to the exit again, you will win the game.

Assumptions:

Having created the tank game already, it was assumed that much of the code could be reused and that we would just have to rearrange the code to the new rules and

setting of our new game. For much of the koala game this was the case and we were able to look at our tank game and do something similar. With that being said, we also knew that this game had to be much cleaner. The tank game, being our first game, obviously had some unconventional methodology, so a lot of the tank game had to be edited before we could start building off of it for the koala game. In fact most of the tank game had to be revised so that it could be reused in koala. This way we could focus on making the koala game clean in terms of movement, user experience and user interface.

Conclusion:

This term project taught us a lot about java application programming, but even more so about working on a team. We started with a blank slate and eventually came up with something that would have been a hit in the late 90's. We ran into many issues with getting settled with github, finding a place to start, dividing up the work and just finding time to do all of this. There was also difference in programming skills which we had to figure out. In terms of code, what we produced came out well if you take into account neither of us had created a java game application before. The games were a challenging project that required pretty much all we have learned thus far in computer science. It was a true test of what skills we have acquired and in the end this was very beneficial to all of our learning experience.

The Following Pages Contain a Breakdown of the Programs Code

Koala_EmptyLvls.java

Summary:

This class is used in the Koala_world class for the time when there are no more levels to the game and ultimately the game has been won.

Data Fields:

Private Thread thread: used in the execution of the program to constantly paint our game for smooth experience
Static JFrame jFrameWindow: the window for the game
BufferedImage buff_Img1 = null: image for the game background
BufferedImage congratulation = null: image for screen when the game has been won

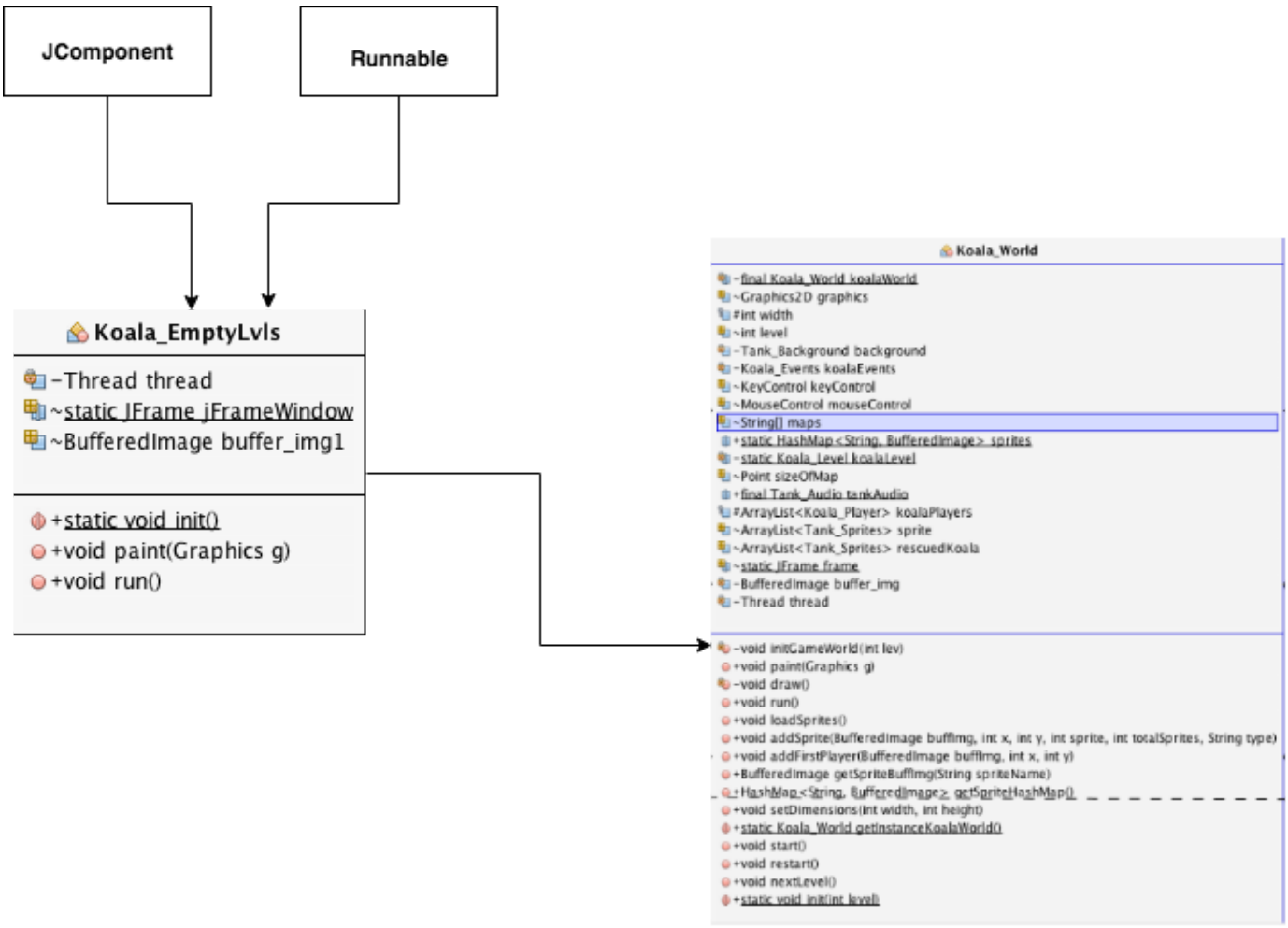
Methods:

static void init(): contains multiple JFrame operations which create the JFrame window. These operations include setting the size, location, visibility, ability to resize and makes it so that when you click exit on the window, the program will stop running. This function is called within the Koala_World class when there are no more levels remaining.

void paint(Graphics g): draws the screen for when the game has been beaten and displays the “congratulation.png” buffered image. Uses a try/catch to assign the BufferedImage variables and uses a Graphic object to draw the images

Void run(): This function maintains the threading and repainting using a while loop and a try/catch statement

Koala_Events Flow



Koala_Events.java

Summary: This class handles the events created by the user's input. Specifically it deals with the actions that take place when a user moves the koala into a given space. This class deals with the game logic for when a koala hits an exit, tnt or a detonator. It also deals with the movement from the directional keys input by the user.

Data Fields:
Public:
Koala_World game: This object of Koala_World koalaGame is used in a logical if statement for when the user clicks the "Restart" button. When the result is true, the koalaGame object will access the restart function in Koala_World and the game will restart. The object is also used to store any given koala if it gets to the exit into a hashmap.

Int [] player1, player2, player3: There are three koala integer arrays to keep track of the three koalas we have in the game. These will be used to store x/y coordinates and to move the koalas

Int saved: an integer that is incremented every time a koala reaches the exit

Methods:

Void process(ArrayList<Koala_Player>, ArrayList<Tank_Game>): calls the move method and the checkCollision method, passes in two array lists

Void pressed(KeyEvent, ArrayList<Koala_Player>): moves the koalas when a directional key is pressed. For each key we have an if statement followed by a for loop. If the key is pressed, the loop will set the direction to true for each koala.

Void released(KeyEvent, ArrayList<Koala_Player>): Similar structure as the pressed() method, but when the directional key is released, this function will set the direction to false.

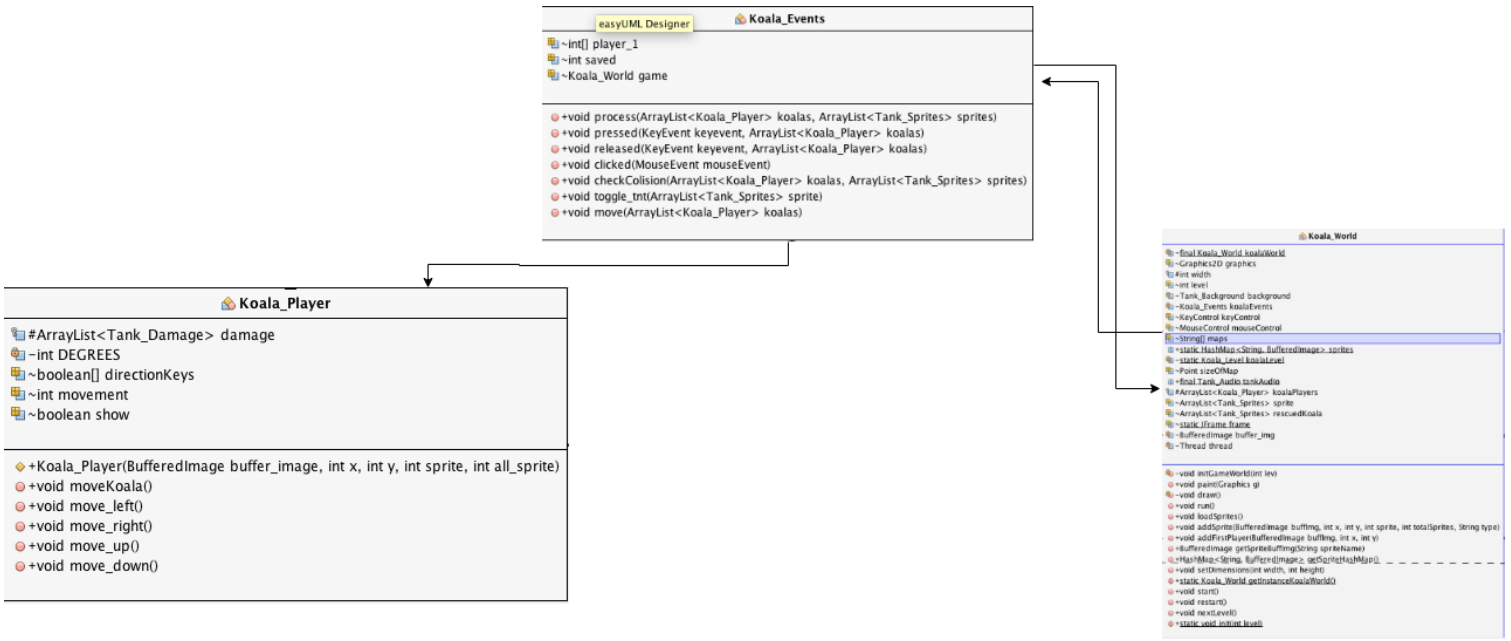
Void clicked(MouseEvent): Large if statement that uses a MouseEvent object to check if the user has clicked the "Restart" button. If true, the retry() method is called and the game will restart.

Void checkCollision(ArrayList<Koala_Player>, ArrayList<Tank_Sprites>): This method checks to see if the koalas collide with any of the four options: the exit, tnt, detonator or another koala. If the koala collides with the exit, the saved variable is incremented, an audio wav is played, the given koala is no longer shown on the map but instead is shown in the rescued koalas section of the screen. When the koala collides with the detonator, the sprite of the detonator being pressed is played and then the toggle_tnt() method is called to explode the tnt. When the koala collides with a tnt, an audio wav is played and the koala is no longer shown. At the end of the method, if the saved variable is equal to three, the game will proceed to the next map and the saved variable reset to zero.

Void tnt(ArrayList<Tank_Sprites>): Removes all the tnt, called in the checkCollision() when a koala collides with tnt.

Void move(ArrayList<Koala_Player>): Gets the x/y coordinates of each koala and calls the moveKoala() method in Koala_Player.java to move the koalas.

Koala_Events Flow



Koala_Level.java

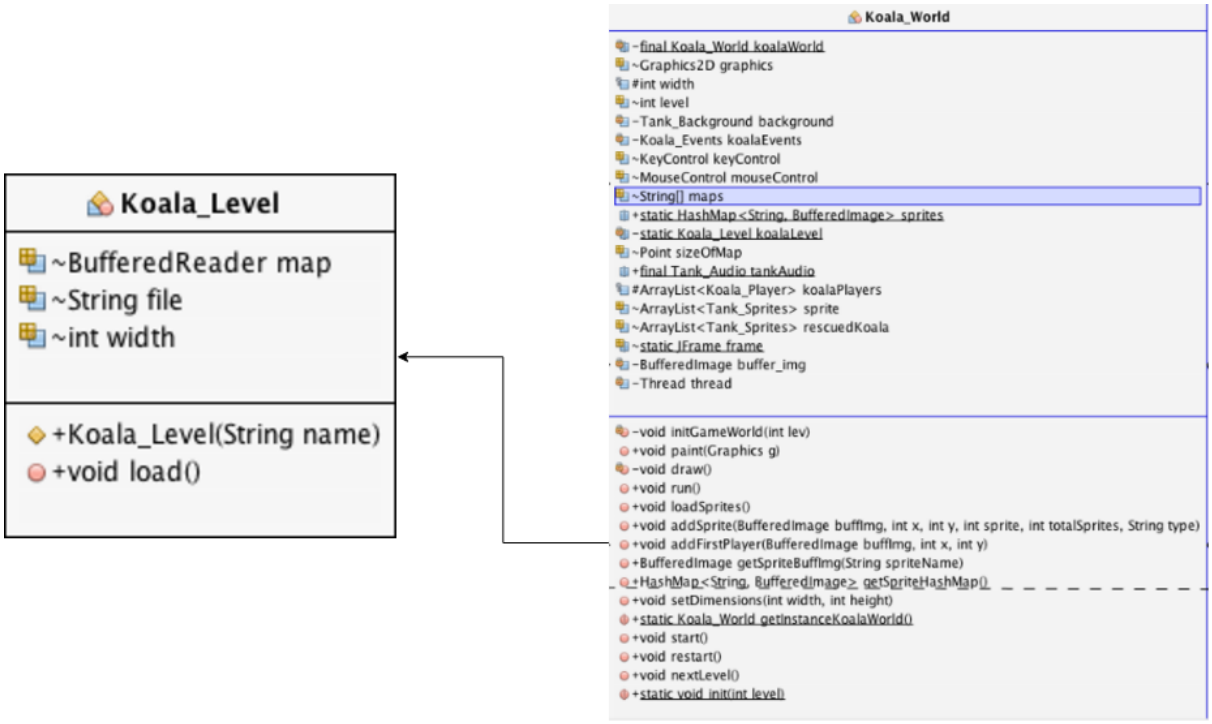
Summary: This file creates the map. Sets up the walls, koalas, tnt, detonator and exit.

Data Fields:
BufferedReader map
String file
Int width, height

Methods:
Koala_Level(String name): this constructor has a try/catch statement which attempts to read in a file which is the form of a .txt file. The file is read by the buffered reader, width and height is adjusted and the map is created.

Void load(): This function loades all the sprites onto the screen. This is achieved through a large switch statement. The switch statements take the values from the text file and if the case is true, the associated sprite will be added. The values are assigned to certain sprites and a sprite hashmap is used to match up the correct sprite that will be added.

Koala_Level Flow



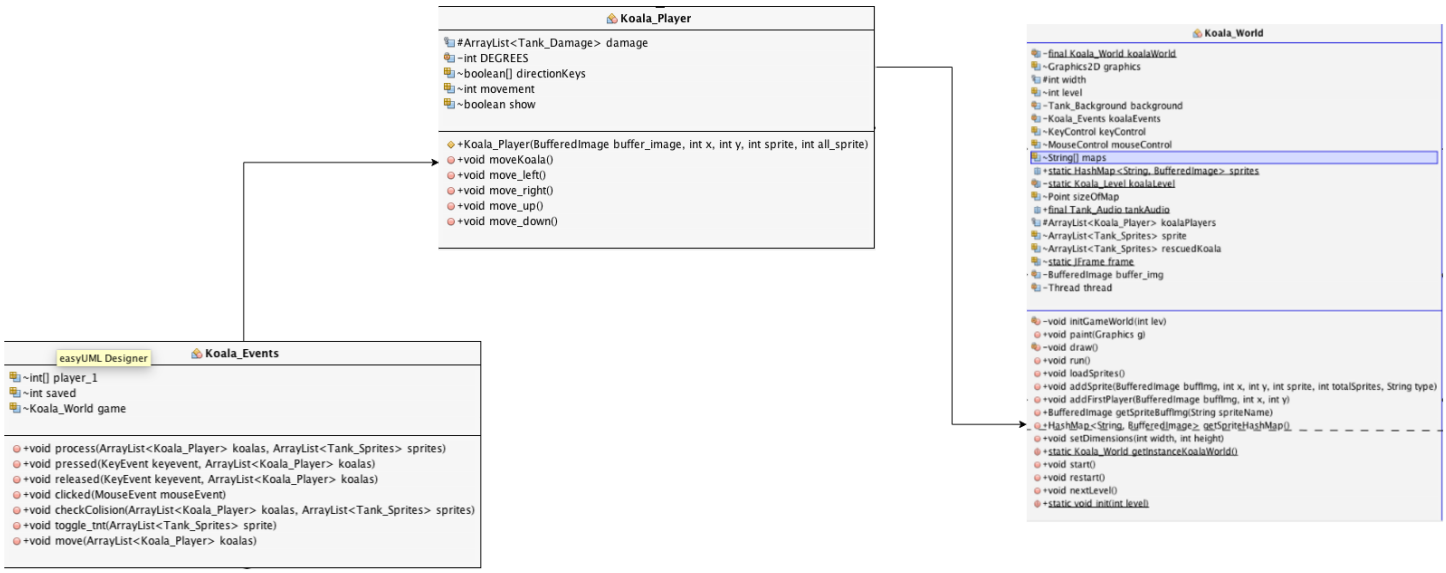
Koala_Player.java

Summary: This file mainly deals with the movement of the Koalas. It takes input data provided from the Koala_Events file and uses it to do the actual moving of the Koalas.

Data Fields:
Protected ArrayList<Tank_Damage> damage
Private final int DEGREES = 6: Degree for the rotations
Boolean[] directionKeys: up, down, left, right
Int movement, playerX, playerY, playerSprite
Boolean show = true

Methods:
Koala_Player(): constructor which initializes all the above data fields
moveKoala(): Method called in Koala_Events file used to move the Koala
moveLeft(): used to move Koala left
moveRight(): used to move Koala right
moveUp(): used to move Koala up
moveDown(): used to move Koala down

Koala_Player Flow



Koala_World.java

Summary: The Koala_World file pulls the program together. It initializes variables used in the methods of other classes, creates the map, provides get and set functions and creates the frame for the game.

Data Fields:

private static final Koala_World koalaWorld = new Koala_World()
Graphics2D graphics
protected int width,height
int level
private Tank_Background background
private Koala_Events events
KeyControl key_movement
MouseControl mouse_movement
String[] maps = new String[2]
public static HashMap<String, BufferedImage> sprites
private static Koala_Level map_level
Point map_size
public static final Tank_Audio tankAudio = new Tank_Audio()
protected ArrayList<Koala_Player> player
ArrayList<Tank_Sprites> sprite
ArrayList<Tank_Sprites> player_saved
static JFrame j_frame
private BufferedImage buffer_img
private Thread thread

Methods:

init_Game(): This init method initializes a lot of the above data fields. The maps are assigned to the proper resources. The sprite hashmap and arraylist are created and loadSprites() is called. Maps get created and events, key and mouse movement are initialized and listeners are added.

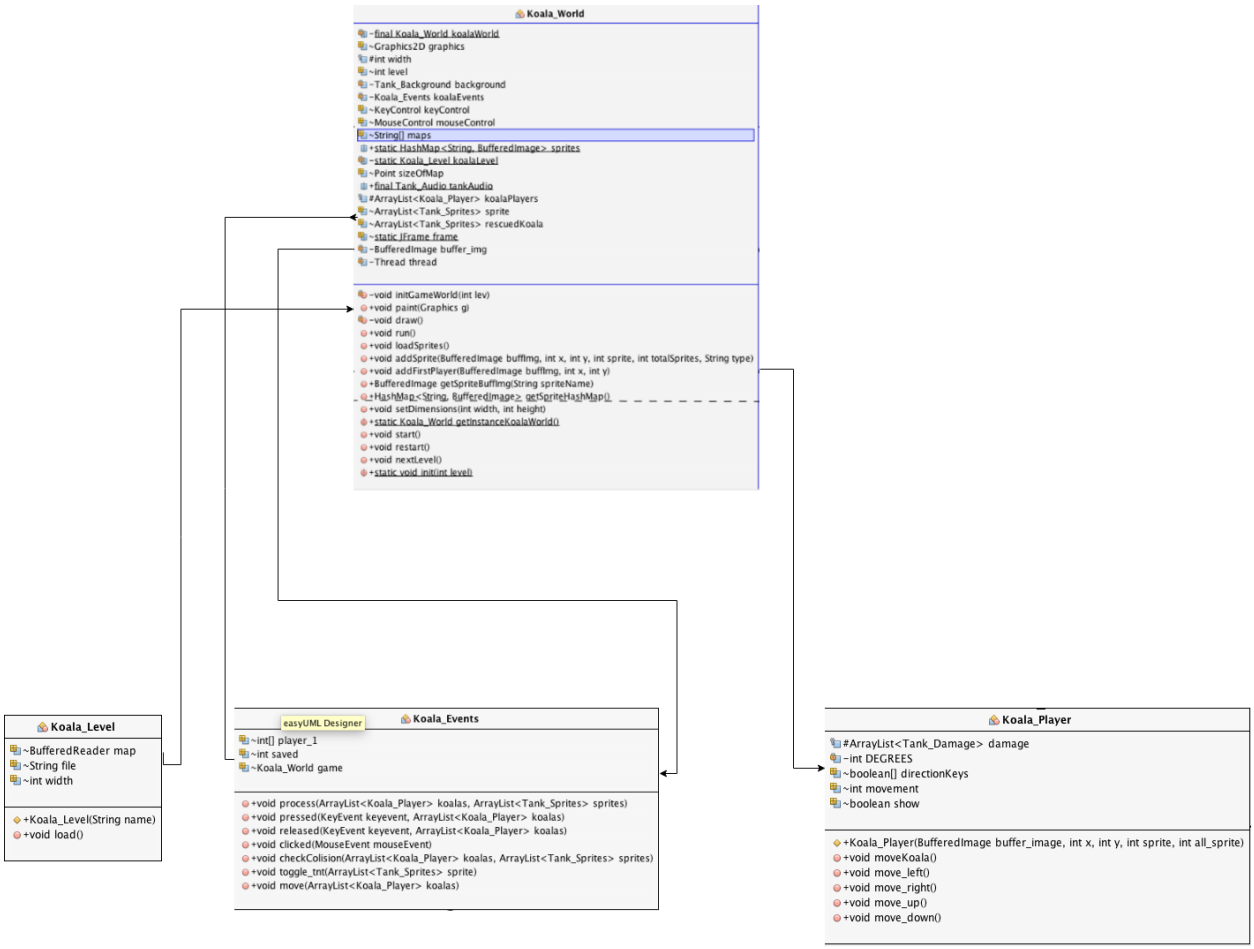
Void paint(Graphics g): Here the bufferedimage is formatted and drawn.

Private void draw(): Called in the paint() method above. Draws the background and also the players and sprites. Accesses the Koala_Events file and the process method when the koalas move.

Void run(): This function maintains the threading and repainting using a while loop and a try/catch statement

Void loadSprites(): Initializes the sprites HashMap with key and resources.
Void addSprite(): Adds new sprite.
Void addPlayer(): Adds new player.
BufferedImage getSpriteBuffImg(): Returns a bufferedimage.
HashMap<String,BufferedImage> getSpriteHashMap(): Returns the sprite hashmap.
Void setDimensions(): Sets game dimensions.
Static Koala_World get_World(): Returns the Koala_World object.
Void start(): Initializes the begins the thread.
Void next_map(): Goes to the next level.
Static void init(int level): Sets up the JFrame and level.

Koala_World Flow



Start_Game.java

Summary: This file contains the main method for the program and starts it all.

Data Fields:
static JFrame window
static String []argument
MouseControl mouse
BufferedImage background = null, title = null, start=null, quit=null
private Thread thread

Methods:
main(): Creates the window, starts the game.
paint(): Creates opening screen which has start and quit buttons.
run(): This function maintains the threading and repainting using a while loop and a try/catch statement

Start_Game Flow