Henry LeCates and Ryan Roberts
PROGRAM SPECIFICATIONS

Part 1:

Our program is a 2d video game with two modes: a level progression mode with four different levels and a two player dueling mode. The program has custom imported graphics we created by hand that gives the game a sense of uniqueness while also maintaining the sophistication and design of the game. The program has levels containing enemy combat, a maze, a puzzle, and a boss with three different attack phases. It also features a two player mode in which you can compete in a game of combat using two different sets of keyboard controls.

The user will provide inputs by clicking keyboard keys to control the character movement and attack. The movement inputs are ‘w’, ‘a’, ‘s’, ‘d’, to control direction of velocity, ‘x’ to stop, ‘e’ to dash forward, and ‘c’ to shoot a projectile. When in dueling mode the respective controls are ‘i’, ‘j’, ‘k’, ‘l’, for direction, ‘m’ to stop, ‘u’ to dash, and ‘n’ to shoot. The direction the projectile is shot has been designed to be in the same way the player is currently moving.

The program is designed to be a challenging and well designed fighting/puzzle video game. The game has enemies in it that actively follow and track the player, firing projectiles to damage the players health. The player must slay these enemies, solve a randomly generated maze, complete a simple puzzle, and then slay a boss with different attack and movement options. This part of the game can be thought of as a campaign mode. The program has many sprites and background elements that are tracked to update collisions.

Part 2:

- **Action:**
    - Semantics:
        - Represents the actions of the player / game when program is run
        - Multiple instances of class, as numerous actions are added to action queue
    - Member Variables:
        - Private String Action
            - Holds the description of the action
            - Private because it is only modified within the class
    - Constructors:
        - Only one, assigns the member variable to the input
    - Methods:
        - Public String toString
            - Returns the member variable
            - Used when printing actions
- **Area:**
    - Semantics:

    - Represents the background of the game screen
    - Composed of an enviro[][]
  - Member Variables:
    - Public Environment [][] enviro;
      - Public because modified / used elsewhere in program
      - Composes the area class, and makes the area broken down into the rectangles
    - Int rows and int columns
      - Public because modified elsewhere in program
      - Can be changed to change the visuals
  - Constructors:
    - Only one, initializes the enviro[][], with double for loops
  - Methods:
    - Public void draw
      - Navigates through the enviro[][] and draws each index
- **Arena:**
  - Semantics:
    - Represents the 1v1 level mode
    - Here there are two players who battle
  - Member Variables:
    - Int height and int width
      - Represents the height and width of the world / screen
    - List<Collectables> powerups
      - Public because modified elsewhere
      - Stores the generated power ups and iterates over them
  - Constructors:
    - Only one, initializes the variables and calls super
  - Methods:
    - Public void update
      - Randomly generates power ups
      - Iterates over power ups and remove them from member variables list if player has collected them
- **Boss**
  - Semantics:
    - Represents the boss that appears in level four of the level progression mode of the game.
    - One instance of the class is created in the World class.
  - Member Variables:
    - Int health
      - Represents the health of the boss

        - Int detectionRadius
            - Represents the radius in which the boss will detect and attack/move towards the player
    - Constructors:
        - Only one, initializes the Boss health and detection radius
    - Methods:
        - Public void draw
            - Draws the boss
        - Public double distanceFromPlayer
            - Returns the distance between the boss and player using the distance formula
        - Public void followPlayer
            - Sets the velocity to move towards the players position
        - Public int getHealth
            - Returns the bosses current health
        - Public void isHit
            - Reduces the health when called (if a projectile collides with the boss)
        - public void handleProjectileRemoval
            - Removes a projectile by setting it to null when called

- **BossHealthBar**
    - Semantics:
        - Draws a bar on the top of the screen representing the current health of the level four boss
        - One instance is created when level four begins
    - Member Variables:
        - Int height and width
            - The height is fixed
            - The width is equal to the health of the boss
        - Pair position
            - The y position is fixed
            - The x position is equal the health of the boss divided by two subtracted by the half the width of the screen
        - Color color
            - The color is yellow by default, orange when the boss is less than 50% health, and red when the boss is less than 25% health
        - Int health
            - The bosses health which is used to calculate the width of the bar

- Constructors:
    - Only one that initializes the initial position of the bar, the height of the bar, the width of the bar, and the color of the bar
- Methods:
    - Public void draw
        - Draws the health bar
    - Public void update
        - Updates the width, position, and color of the bar all depending on the health of the boss
        - Checks if the boss's health is depleted and teleports the boss off the screen when it is
    - Public boolean isDead
        - Returns true if the boss has no more health
        - Returns false if the boss still has health
    - Public Color getCurrentBarColor
        - Takes the health and returns the appropriate bar color based on the health.

- **BossTimer**
    - Semantics:
        - Represents a sliding cube with a varying x position that changes the attack phase of the level four boss
        - The color of the cube will change which will be used as an input for the level four boss to determine which attack phase it is in
    - Member Variables:
        - Color color
            - Used to represent the phase of the boss
        - Int phase
            - Integer value determined by color
        - Pair position
            - Represents the position of the timer
        - Pair velocity
            - Represents the speed of the timer, can be changed to +/= phase change speed
        - Int height and width
            - Dimensions of timer
    - Constructors:
        - One constructor that initializes height, width, position, velocity, color, and phase to preset values that all be updated (besides height and width)
    - Methods:

        - Public void draw
            - Draws the timer
        - Public void update
            - Changes the color to yellow, orange, and red using the current x position and inequalities
            - Reverses the velocity when the timer hits specific x values marking the border of the screen
            - Adds the velocity to the current position to cause movement
        - Public int getBossPhase
            - Returns an integer 1, 2, 3, or 0 representing the phase of the boss
            - Uses the current timer color to determine which int to return

- **Button:**
    - Semantics:
        - Abstract class which defines method for the buttons in the title Screen
    - Member Variables:
        - Color color and String string
            - Used to draw before addition of graphics
            - Stores what the button is supposed to do
        - Pair position
            - Stores where the button is located on screen
    - Constructors:
        - Only one, initializes variables to input parameters
    - Methods:
        - Abstract public void collision, Abstract public void handleCollision, Abstract public void draw
            - All methods are overridden in extensions of button

- **Collectables**
    - Semantics:
        - Represents anything that the player is able to pick up and use
        - An abstract class used as basis for extensions
    - Member Variables:
        - Pair Position
            - Used to store x,y position
        - Double width, double height, Color color
            - Used in draw method and collision detection
    - Constructors:
        - Only one, initializes the variable given input parameters
    - Methods:

        - Public double distance()
            - Uses distance formula to find distance
        - Public boolean collisionPlayer()
            - Checks if player has collided with collectable
        - Abstract void draw, abstract void function
            - Defines methods which must be defined in implementation of class
- **Enemy**
    - Semantics:
        - Represents an enemy with a certain health that fires projectiles towards the player
        - Multiple instances are created in World for level one
        - Used as a blueprint for the Boss class
    - Member Variables:
        - Int health
            - Represents the health of the Enemy
        - Int detectionRadius
            - Represents the radius in which the Enemy will detect and attack/move towards the player
        - Long previousShootTime
            - Represents the time before the last enemy projectile is fired
            - Used to control rate of projectile shooting
    - Constructors:
        - Only one, initializes the Enemy health and detection radius
    - Methods:
        - Public void draw
            - Draws the boss
        - Public double distanceFromPlayer
            - Returns the distance between the boss and player using the distance formula
        - Public void followPlayer
            - Sets the velocity to move towards the players position
        - Public void isHit
            - Reduces the health by 50 when called (if a projectile collides with the boss)
        - public void handleProjectileRemoval
            - Removes a projectile by setting it to null when called

- **Environment:**
    - Semantics:
        - Represents each individual rectangle drawn in the background

        - Works with area class to draw background
    - Member Variables:
        - Pair position
            - Stores x,y position
        - Int height, int width, Color color
            - Used in collisions detections and drawing
        - Boolean solid
            - Used in level 2 (maze), allows for specific environments to be designated as walls
    - Constructors:
        - Only one, initializes variables given input parameters
    - Methods:
        - Public void setColor
            - Allows environments to be a different color and accessed another classes
        - Public void setSolid
            - Allows environments to designated as walls
        - Public boolean isSolid
            - Allows for checking if a wall
        - Public void draw
            - Draws a simple rectangle

- **Explorer:**
    - Semantics:
        - Houses the main elements of the program
            - ie runnable, paintcomponent, keyBoard inputs, and the main method
    - Member Variables:
        - Public static final int WIDTH, public static final int Height, public static final int FPS
            - Constants in the program that help to define the world bounds and are never changed, only passed as parameters
    - Constructors:
        - Only one, initializes a new world and adds the necessary components so that the program will run
    - Methods:
        - KeyBoard Inputs
            - A series of methods, which must be implemented
            - The main one used is keyPressed which handles all player movements etc

        - Public void paintComponent
            - Handles all drawing of world
        - Class runner
            - Handles all update methods of world

- **FinalScreen:**
    - Semantics:
        - Displays the final Screen when player wins or dies
    - Member Variables
        - Int height, int width
            - Used to represent how large the screen is
        - Boolean win
            - Determines what message is given and what colors are drawn
    - Constructors:
        - Only one, super call, initializes variables
        - Uses the player health to determine if the player won or lost
            - Using this sets the background accordingly
    - Methods:
        - Public void update
            - Empty method
        - Public void draw
            - Uses the win boolean to draw a game over or nothing
            - Also draws prompt to begin again

- **GenericQueue:**
    - Semantics:
        - Creates a user defined queue using generics class
        - Is utilized in the program to implement powerups and writing the actions
    - Member Variables:
        - Private QueueNode<T> next
            - Private because only modified in in the class
            - Represents the pointer to the front of the queue
        - Private QueueNode<T> previous
            - Private because only modified in in the class
            - Represents the pointer to the rear of the queue
    - Constructor:
        - Only one, initialize member variables to null, as queue would be empty
    - Methods:
        - All public because accessed in other classes
        - Public void addElement

        - Allows program to add elements to queue
    - Public T pop
        - Removes and returns the element from the front of queue
    - Public boolean isEmpty
        - Returns boolean checking if the front of the queue is null
        - Used in error checking and prevention

- **Invulnerable:**
    - Semantics:
        - Represents the invulnerable powerup in the arena mode
        - Multiple instances stored in a list
    - Member Variables:
        - None, all are contained in parent class
    - Constructor:
        - Only one, super call, and initializes variables
    - Methods:
        - Public void function
            - Calls methods in player which modify some aspect of player
        - Public void draw
            - Uses imported image to draw

- **Key**
    - Semantics:
        - Represents a key sprite that spawns in when certain level objectives are met (ex. Killed all the enemies)
        - This key allows you to progress to the next level by moving the player to the bounds of the game (off the screen)
    - Member Variables:
        - Color color
            - Default color set to blue
    - Constructors:
        - Only one, initializes the color and calls super
    - Methods:
        - Public void function
            - Used in other collectables to activate their functions, not needed in Key as it was replaced by a boolean to track if the player has the key
        - Public void update
            - Checks if the player collides with the key
        - Public void draw

        - Draws the key
- **LevelOne**
    - Semantics:
        - Represents the first level of the level progression side of the game
        - Spawns in a randomly generated number of enemies that the player must shoot projectiles at to spawn in the key and progress to the next level
    - Member Variables:
        - Int totalEnemies
            - Represents the randomly generated number of enemies
        - Boolean keyGenerated
            - Represents whether or not the key is generated
        - Key key
            - The key collectable that is used to move to the next level once the level objective is complete
        - Private boolean resetPlayerPosition
            - Keeps track when the player's position is reset to the center of the screen with no velocity
    - Constructors:
        - One constructor, supers the area constructor to create a 15x15 background of green rectangles
        - Creates 5-15 enemies and then adds the to World
        - Sets the player position to the center with no velocity
    - Methods:
        - Public void update
            - Checks if the player position should be reset
            - Makes all enemies follow the player if in a certain radius of player
            - Updates all enemies
            - Checks if all enemies are killed to know when to generate the key (and generates the key)
        - Public void draw
            - Draws the environment
            - Draws the enemies
            - Draws the key
            - Draws text telling the player what to do (Defeat enemies and press 'c' to attack)

- **LevelTwo**
    - Semantics:
        - Represents the second level of the game
        - Creates a randomized maze which player navigates through

- Member Variables:
    - All private because only modified in class and shouldn’t be changed elsewhere
    - Private boolean reset Playerhealth
        - Allows level to track if the player health has been reset
    - Private boolean resetPlayerPosition
        - Allows for checking if player position has been randomized to a path and player velocity set to 0
    - Private boolean resetPlayerSize
        - Allows to check if the player size has been changed so that the player can navigate the maze without changing collision detection or size of maze
    - Private Random rand
        - Level two random, used in randomizing directions
    - Private boolean[][] maze
        - A double array of booleans which represents the walls and paths of the mazer
    - Private Pair[] directions
        - [] of all possible directions that can be traveled from one cell
    - Private Stack<Pair> stack
        - Data structure used in the backtracking portion of the maze generation algorithm
    - Private List<Pair> availablePaths
        - List of the available paths, used in randomly generating the player and key positions
- Constructors:
    - Only one, super call, then initializes maze as blank slate, and then creates maze
    - Then sets colors of walls to be random shade of gray, then uses available paths to randomly generate a key
- Methods:
    - Public void createMaze
        - Iterates through the stack and looks at the current cell
        - Then if the cell has a valid neighbor adds to paths
        - If no valid neighbors, backtracks
    - Public Pair getRandomNeighbor
        - Looks at the cells in random direction and checks if the cell meets conditions for valid neighbor
    - public void shuffle
        - Uses the fisher yates algorithm to randomly shuffle the directions[]

        - Public boolean valid Neighbor
            - Has a conditional which checks the validity of the neighbor
                - ie if cell is outside bounds, or already been looked at
        - Private void updateAvailablePath
            - Looks at a cell and the surrounding neighbors, to see which neighbors are paths

- **LevelThree**
    - Semantics:
        - Represents the third level of the game
        - A simple puzzle with 9 different nodes that can be toggled by shooting a node → all nodes must have the correct color that is randomly chosen before the key is generated (to determine whether or not a node is the correct color you must shoot it and then watch the change in node % correct on the top of screen)
    - Member Variables:
        - Boolean keyGenerated
            - Represents whether or not the key is generated
        - Key key
            - The key collectable that is used to move to the next level once the level objective is complete
    - Constructors:
        - One constructor, supers the area constructor which creates a 48x48 area
        - Initializes the key
    - Methods:
        - Public void update
            - Checks for node values to know when the puzzle is solved and the key should be generated
        - Public void draw
            - Draws the environment/background
            - Draws the key when it is generated
            - Draws nodes 1-9 and draws text to represent the % of nodes that are correct

- **LevelFour**
    - Semantics:
        - Represents the fourth level of the game
        - Has a boss with different phases and a health bar, to finish the game you must shoot the boss with projectiles until it has no more health

        - The boss will shoot projectiles and follow the player with the same methods that the enemies used
    - Member Variables:
        - Boolean keyGenerated
            - Represents whether or not the key is generated
        - Key key
            - The key collectable that is used to move to the next level once the level objective is complete
    - Constructors:
        - One constructor, supers the area constructor to create a 24x24
        - Initializes the key
    - Methods:
        - Public void draw
            - Draws the environment/background
            - Calls deleteNode to remove nodes 1-9 from the previous level
            - Draws the key when the boss has no more health
            - Draws the Boss, BossTimer, and BossHealthBar
        - Public void update
            - Updates the health bar to reflect the current boss health
            - Updates the boss to follow and attack the player
            - Updates the timer to move left and right to change the boss phases
            - Includes all the attack and movement behavior that is determined by the current boss phase
                - Phase 1 (yellow) → follows player at a somewhat varying speed and shoots basic projectiles
                - Phase 2 (orange) → follows player at a slower constant speed and shoots tons of spread out white projectiles that cover large areas
                - Phase 3 (red) → follows the player at a varying fast speed and shoots fast little red projectiles

- **LevelStart**
    - Semantics:
        - Represents a button that will start the level progression mode of the game when the player moves to the button
    - Member Variables:
        - None
    - Constructor:
        - One constructor, supers the button constructor to create a button with a given color, string, and position

- Methods:
    - Public void function
        - Increases the levelIndex by one
    - Public void collision
        - Checks if the player collides with where the level button is, if the player does collide the levelIndex will increase by one
    - Public void draw
        - Draws the levels button image from LevelsButton.png

- **Node**
    - Semantics:
        - Represents the nodes used level three of the game
        - Each node has a random correct value, either 1 or 2. Being shot by a projectile changes their value from either 1→2 or 2→1. If their color is matched to their correctly given random value, the node percent will increment by 11.11 %. Once the node percent is equal to 99.99 %, the key will spawn.
    - Member Variables:
        - Pair position
            - Used to place the nodes on map for level three
        - Color color
            - Used to visually depict the current value the node has (either white or black)
        - Int correctColor
            - The value the node needs to be to add 11.11% to the total node percent, 1 represents white and 2 represents black
        - Int currentColor
            - The current value the node has, 1 represents white and 2 represents black
        - Int height
            - Represents the node height
        - Int width
            - Represents the node width
    - Constructor:
        - One constructor, initializes the position, currentColor, and correctColor
        - Sets the height and width to 50
        - Changes the visual color of the node based on the value of currentColor (every node starts as white)
    - Methods:
        - Public void changeNode

        - Switches the current color of the node by checking for the currentColor and switching from either 1 to 2 or 2 to 1
    - Public void draw
        - Draws the node
    - Public void deleteNode
        - Draws the node but with no width and height so that it is not visible
    - Public static boolean checkNode
        - Sees if the currentColor int value of the node is equal to the correctColor int value, returns true if they equal
    - Public static double nodePercentCorrect
        - Uses the above method to create a double value representative of how many nodes currently have correctColor == currentColor (ex. 55.55% = 5/9 nodes, 99.99% = 9/9 nodes)
        - This double value is printed on the screen to give the player enough information to solve the node puzzle

- **NodeButton**
    - Semantics:
        - Collects all the data from the nodes and draws the % of nodes correct to the screen so the player can have enough information to solve the puzzle
        - Originally was a physical button but got rid of physicality to improve the ease of playability
    - Member Variables:
        - None
    - Constructor:
        - One constructor to create the button, initializes variables but they are no longer needed in the game as the button is not a physical object
    - Methods:
        - Public double isPressed
            - Uses static method nodePercentCorrect of Node to return a double value (11.11, 33.33, 99.99, etc.) for the % of nodes that are correct
        - Public void drawNodeResult
            - Draws text to the screen using the % of nodes that are correct found in the isPressed method
        - Public void draw
            - Draws the node button

- **Pair**
    - Semantics:

        - Allows us the store a pair of values in an object
        - Typically a position or velocity
        - Many instances as most of the class have a pair position
    - Member Variables:
        - Double x and Double y
            - Public because they are commonly accessed from other classes
            - Stores the passed x,y value
    - Constructor:
        - Only one, initialize member variables to passed value
    - Methods:
        - Public Pair add, Public Pair times, Public Pir divide, Public void flipX, Public void flipY
            - All methods relate to a mathematical operation
            - Used in calculating velocities, accelerations etc

- **Person**
    - Semantics:
        - Used to create a person that will be extended in the following classes: Player, Enemy, Boss
        - This class will be abstract to permit us to work with polymorphism (does things like making collision detecting easier) across the Player, Enemy, and Boss class
    - Member Variables:
        - Pair position
            - Represents the current position of the person
        - Pair velocity
            - Represents the current velocity of the person
        - Double width
            - Represents the width of the person
        - Double height
            - Represents the height of the person
        - Double radius
            - Represents half the height of the person
        - Color color
            - Represents the color of the person
        - Int health
            - Represents the health of the person
        - Projectile projectile
            - Creates a projectile object that is later sprited when a person attacks

        - Long previousProjectileTime
            - Stores the time since the last projectile was created, used to change rate of fire/attack
    - Constructor:
        - One constructor, initializes position, velocity, height, width, color, health, and sets radius equal to half of height
    - Methods:
        - Public void update
            - Updates the person's position based on their velocity
        - Public void setPosition
            - Changes the person’s position to a given Pair
        - Public void setVelocity
            - Sets the person’s velocity to a given Pair
        - Abstract public void draw
            - Provides a template to be implemented in classes that extend Person for drawing their sprites

- **Player**
    - Semantics:
        - The Player class extends Person and represents the main character/sprite of the game
        - This class will store methods for player position and movement like keeping the player in the bounds of the screen.
    - Member Variables:
        - Long previousDamageTime
            - Stores how many milliseconds since last damaged
        - Long previousDashTime
            - Stores how many milliseconds since last dash
        - Boolean key
            - Determines whether or not the player has the key
        - Int dashDistance
            - A predetermined distance the player moves when inputting the dash control
        - Boolean dashUp
            - Used to determine which way the player dashes based off the current player velocity
        - Boolean dashDown
            - Used to determine which way the player dashes based off the current player velocity
        - Boolean dashLeft

        - Used to determine which way the player dashes based off the current player velocity
    - Boolean dashRight
        - Used to determine which way the player dashes based off the current player velocity
    - List<Projectile> projectiles
        - Allows for multiple projectiles
    - Int projectileCooldown
        - Changes to determine whether or not the player is allowed to shoot another projectile
    - GenericQueue<Collectables> power ups
        - Powerups that can be collected in the two player arena game mode
    - Long projectileStartTimer
        - Used to speed up projectile firing rate after collecting a powerup
    - Boolean invulnerable
        - Determines whether or not the player can be hit by a projectile and lose health
    - Long invulnerable Start Time
        - Used for power ups to track how long since the powerup has been active
- Constructor:
    - One constructor, initializes level, key, dashUp, dashDown, dashLeft, dashRight, dashDistance, projectile, projectiles, projectileCooldown, powerups, invulnerable
- Methods:
    - Public void update
        - Supers the update method in Person
        - Checks if any power ups are active
    - Public boolean outBounds
        - Checks all outBounds methods and returns true if any are true
    - Public boolean outBoundsUp
        - Returns true if the player moves out of the screen bounds up
    - Public boolean outBoundsDown
        - Returns true if the player moves out of the screen bounds down
    - Public boolean outBoundsLeft
        - Returns true if the player moves out of the screen bounds left
    - Public boolean outBoundsRight
        - Returns true if the player moves out of the screen bounds right
    - Public void dash
        - Allows the player to travel a set distance instantaneously

        - Public void isHit
            - Subtracts 10 health if the player is hit
            - Checks if the player has any health left and ends the game when health <= 0
            - Uses previousDamageTime to add a cooldown to being attacked
            - If the invulnerable powerup is activated (invulnerable = true), then the player can not be hit
        - Public void newProjectile
            - Returns if a new projectile can be made by checking the cooldown time
        - Public void addPowerup
            - Adds a powerup to the queue
        - Public void usePowerup
            - Uses a powerup
            - Removes the powerup from the queue
        - Public void healthBoost
            - Adds 10 health to the player
        - Public void projectileSpeedup
            - Starts the powerup timer
        - Public void invulnerablePowerup
            - Starts the powerup timer
        - Public void draw
            - Draws an image of the player (player.png)
            - Draws a visual effect if invulnerable
            - Draws the players hearts (heart.png)

- **Projectile:**
    - Semantics:
        - Represents a projectile fired by a person
        - Many instances
    - Member Variables:
        - Pair position
            - Stores where projectile is
        - Pair velocity
            - How fast projectile moves
        - Double deAcceleration
            - Represents at what rate the projectile slows
        - Color color, Int height, int width
            - Used in drawing the projectile
        - Int designation

        - Tracks what type of person fired projectile
    - Person Shooter
        - Similar to designation, but stores who fires projectile
    - Boolean Collided
        - Tracks if the projectile has collided with another object
  - Constructor:
    - Only one, initialize member variables to input parameters
  - Methods:
    - Public void updateProjectile
        - Moves and slows projectie
        - Calls collision detection for projectile
    - Public boolean outOfBounds
        - Checks if projectile is out of bounds
    - Public boolean Stopped
        - If projectile moves at below certain speed, return true
    - Private void checkPlayerCollision
        - Iterates through players and checks collisions
    - Private void checkEnemyCollision
        - Iterates through enemies and checks collisions
    - Private boolean collision
        - Checks if projectile overlaps with any of drawn sprite
    - Private void checkBossCollision
        - Checks for collision with boss and handles accordingly
    - Private void checkNCollision
        - Calls nodeCollision
    - Private boolean nodeCollision
        - Calculates the distance between projectile and node
    - Public void draw
        - Draws projectiles
- **ProjectilePowerup**
  - Semantics:
    - Represents the projectile powerup in the arena mode
    - Multiple instances stored in a list
  - Member Variables:
    - None, all are contained in parent class
  - Constructor:
    - Only one, super call, and initializes variables
  - Methods:
    - Public void function
        - Calls methods in player which modify some aspect of player

        - Public void draw
            - Uses imported image to draw
- **HealthBoost**
    - Semantics:
        - Represents the health powerup in the arena mode
        - Multiple instances stored in a list
    - Member Variables:
        - None, all are contained in parent class
    - Constructor:
        - Only one, super call, and initializes variables
    - Methods:
        - Public void function
            - Calls methods in player which modify some aspect of player
        - Public void draw
            - Uses imported image to draw
- **QueueNode**
    - Semantics:
        - Represents a node in a generic queue data structure
        - Many instances
    - Member Variables:
        - T value
            - Represents the data stored in the node
        - QueueNode<T> next
            - A reference to the next node in queue
    - Constructor:
        - Only one, allows creations of new node with specified value
    - Methods:
        - None
- **ReadImages:**
    - Semantics:
        - Reads and stores images that are used to draw the sprites
        - One instance
    - Member Variables:
        - Public Image title, Public Image levelsButton, Public Image arenaButton, Public Image key, Public Image player, Public Image enemy, Public Image heart, Public Image invulnerablePowerup, Public Image projectilePowerup, Public Image boss, Public Image gameOver
            - All public because must be accessed to draw
    - Constructor:
        - Only one, initializes the images by reading from a file

        - Uses IOException to catch any errors
    - Methods:
        - None
- **TitleScreen:**
    - Semantics:
        - Represent the title screen
        - Allows the player to interact with the buttons and choose which mode is played
    - Member Variables:
        - Int height, int width
            - Represent the bounds of the world
        - twoPlayerStart twoPlayerStart, levelStart levelStart
            - Types of buttons which both have functions when player collides with them
        - Public boolean isButtons
            - Determines if the level has buttons
        - Public buttons[]
            - Allows for easier iteration over buttons
    - Constructor:
        - Only one, super call
        - Initializes the buttons for the different modes
    - Methods:
        - Public void update
            - Iterates over buttons and checks collisions with player
            - The player acts as a cursor
        - Public void draw
            - Draws the environments
            - Call draw methods of buttons
- **twoPlayerStart**
    - Semantics:
        - Represents a button that will start the arena mode of the game when the player moves to the button
    - Member Variables:
        - None
    - Constructor:
        - One constructor, supers the button constructor to create a button with a given color, string, and position
    - Methods:
        - Public void function
            - Calls world updateTwoPlayer method

    - Public void collision
        - Checks if the player collides with where the buttons is, if the player does collide the function will be called
    - Public void draw
        - Draws the levels button image from ArenaButton.png
- **World**
    - Semantics:
        - Stores all the main aspects of the game, and controls the update and draw functions of the program
    - Member Variables:
        - ReadImages image;
            - Creates the class which stores and reds the images from files
        - Random random
            - Creates a random which is used throughout for consistent randomness
        - Int seed
            - Used to create random
        - Boolean gameOver
            - If player dies sets to true, indicating things should be changed in levels
        - GenericQueue<Action> actionQueue
            - Stores all the game actions
        - Int height, int width
            - Controls how large screen is
        - Boolean twoPlayer
            - If user selects arena button this is set to true changing aspects of world
        - Player player, player player2
            - Stores all data related to player
        - List<player> players
            - Allows us to easily iterate through players
        - Int playeVel
            - Determines at speed in which player moves
        - Level[] levels
            - Stores all levels in game
        - Int levelIndex
            - Controls where / what we are acting on in levels
        - Node node1 - 9
            - Used in level three node puzzle
        - Node Button button

        - Used in level three node puzzle
    - Boss finalBoss, BossTimer timer, Boss Health bar health bar
        - Stores all data concerning boss used in level four
    - List<Enemy> enemies
        - Iterates through and controls updating / drawing
    - Int enemies killed
        - Used in level one to generate key
    - List<Projectile> projectiles
        - Iterates through and controls updating / drawing

- Constructor:
    - Only one, initializes each member variable to new objects, or sets int / boolean to their according value
- Methods:
    - Public void updateAll
        - Calls all update methods in world
    - Public void drawAll
        - Calls all draw methods
    - Public void updateTwoPlayer
        - If user selects arena mode, changes certain values in world to accommodate
    - Public void updatePeople
        - Calls update method for enemies and players
    - Public void update Player
        - Updates player and handles out of bounds
    - Public void draw Player
        - Iterates through players and draws the player
    - Public void updateEnemies
        - Iterates Enemies and updates them
        - Removes the ones that are dead
    - Public void updateArea
        - Updates levels at levelIndex
    - Public void drawArea
        - Draw levels at levelIndex
    - Public void updateProjectiles
        - Iterates through and updates
        - Iterates through and removes the ones that have collided or stopped
    - Public void addProjectile
        - Adds a specific projectile to list
    - public void removeProjectile

        - Removes specific projectile from list
    - Public void drawProjectiles
        - Iterates through projectiles and draws
    - Public void gameOver
        - Calls print actions and sets boolean
    - Public void newAction
        - Adds an action to action queue
    - Public void printActions
        - Create s writer which writes the action queue to a file

Part 3: Interaction of Classes

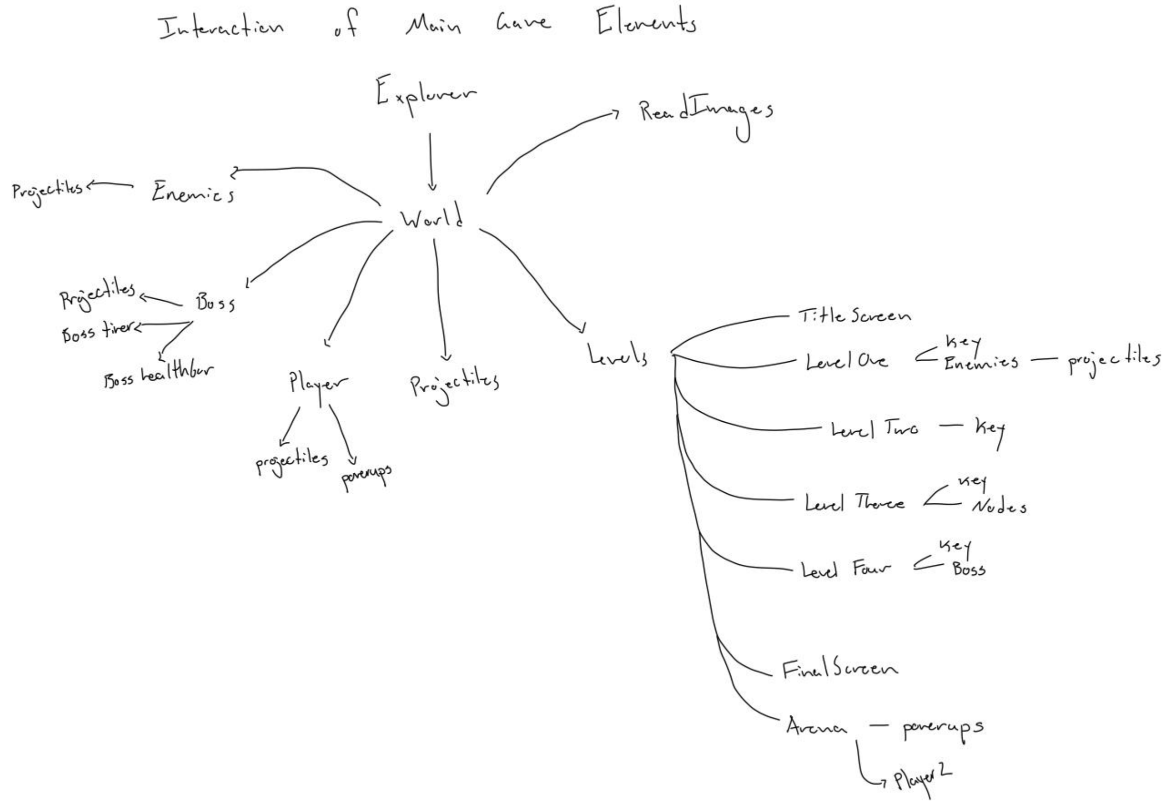

Image shows basic interaction of the core game elements, ie what each class stores and what classes are stored in what classes. In general World is linked to all classes, with it storing the player, levels, enemies list, projectiles list and the boss elements. The projectiles list is stored in the world, however, the projectiles are created in the various person classes. The keys are all stored in the respective levels, as they behave slightly differently. Then the area and environment class (not shown in image) are incorporated into the Level classes. The buttons are used in the title screen. In general our program follows a hierarchical structure where the world stores smaller classes, and those smaller classes store smaller classes, and so on, with each smaller class having an effect on where it is stored.