ARTEMIS DASH

TECHNICAL MANUAL

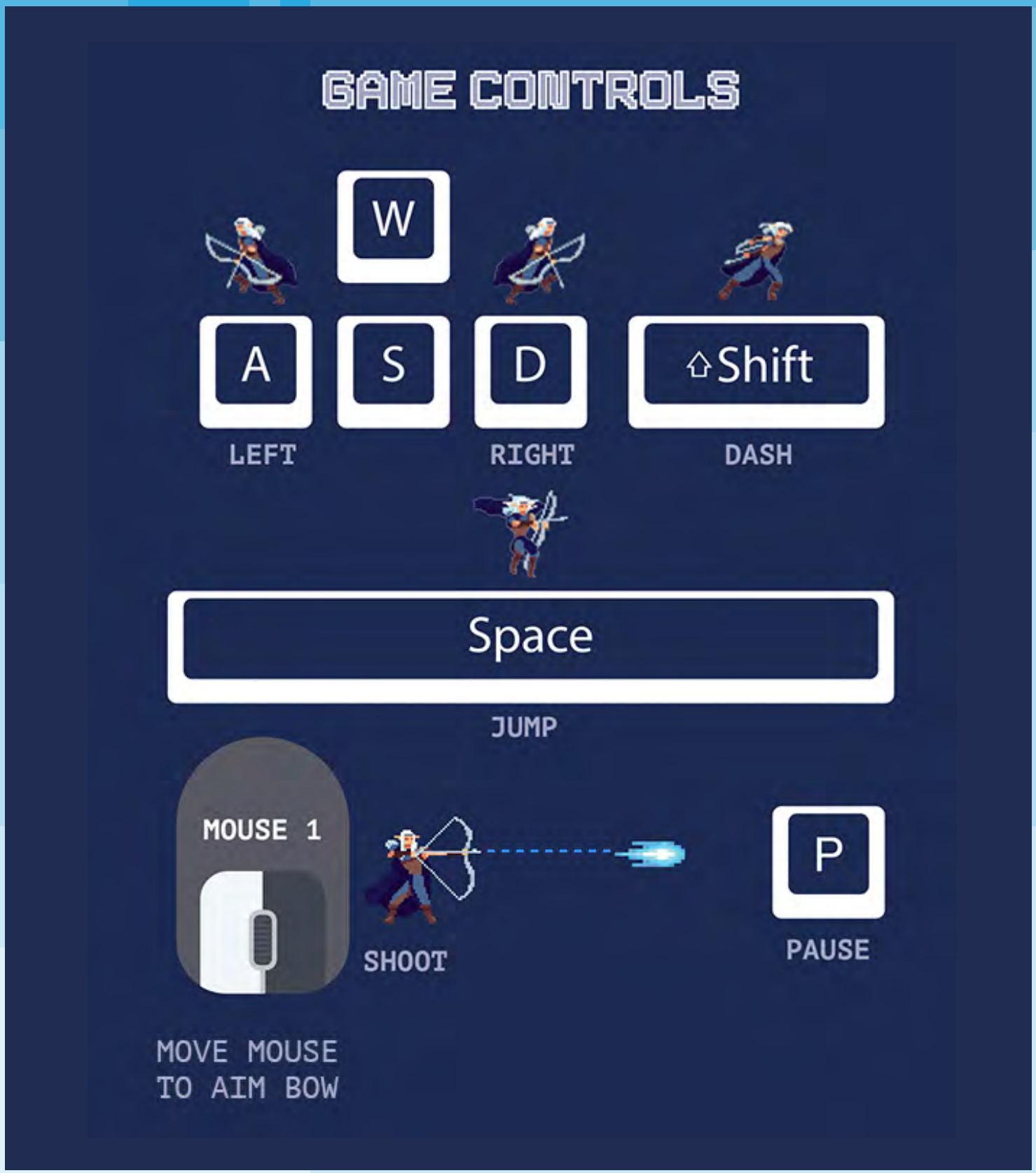
EMBARK ON AN ADRENALINE-FUELED RESCUE MISSION WITH ARTEMIS DASH!

Join Artemis on a picturesque picnic turned perilous quest as her companion, Callisto, is suddenly kidnapped by Hades, vanishing into the Nea Kameni volcano! With no time for questions, Artemis leaps into action, diving after her friend and into the belly of the volcano. Get ready to plunge into the fiery depths of the Nea Kameni in Artemis Dash!

Inspired by classic side-scrollers like Sonic and Mario, as well as the high-stakes gameplay of modern classics like Celeste and Super Meat Boy, this high-speed, action-packed adventure puts you in control. With tight platforming mechanics, lightning-fast dashes, and your trusty bow, you'll brave the lava and face more than just hot magma lurking in the darkness of the caves. Gather your wits, aim true, and dash through the hazards to save your friend from Hades! Are you ready to become the hero Santorini needs?

GETTING STARTED

To play this game, simply run the .jar file as you would any other executable!



GETTING STARTED continued

HUD

SCORE:
KILL ENEMIES TO INCREASE YOUR SCORE!

ENEMIES LEFT:
HOW MANY MINIONS ARE LEFT. KILL THEM ALL TO PROGRESS!

HEALTH:
YOU HAVE 2 HITS BEFORE YOU PERISH. MAKE 'EM COUNT!

SKELETON

(100 pts)

This is a skeleton, one of Hades' risen minions.
Patrolling his volcano fortress,
these undead may look weak, but don't be fooled!
While they are all bones, they still pack a wallop.
Get too close, and you'll feel the sting of its steel!

HEALTH - 1 Hit

SKELETON KING

(2000 pts)

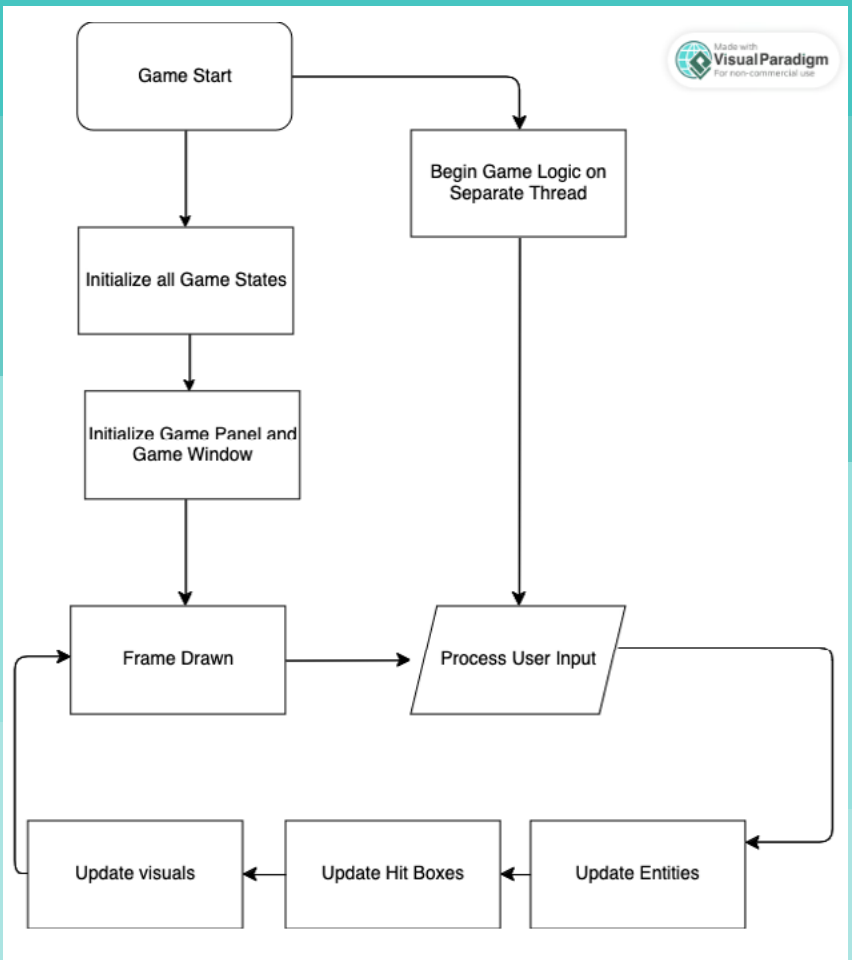
The king has arrived!
While Hades can't micromanage everything,
he delegates his rule of the undead to this spooky king.
You can spot him by his gold crown but be warned!
He is tougher than the rest of his minions and
can put up more of a fight. Be careful!

HEALTH - 3 Hits

ARCHITECTURE OVERVIEW

High-Level Architecture

The core loop revolves around the actions and processes between each frame draw.



Once started, the game initializes every game state, starting with the main menu. It then sets up the game panel and begins the game logic on a separate thread. From here out, the process is much the same for every interaction the user can have with the game, from the GUI to dashing. In between every frame draw, the game checks for inputs through the currently selected game states update method. It first checks if any input has been pressed from the user. If the user input was to jump by pressing spacebar, or click to select a menu item, the action would register at this stage, before moving on to the rest of the game. Speaking only to the “Playing” game state, using the substantial update function, It first checks to see if the level is currently running without being completed or paused. If not, it continues with each entity within the current game state checking for any changes in its environment / hitbox, starting with the player.
After everything is checked to be okay. The new frame draws with the parameters now ready for the next input.

ARCHITECTURE OVERVIEW continued

At the highest level, Artemis Dash is a very fast slideshow produced through the Graphics AWT, with inputs and game logic processed in between slides! In a more serious sense, the core functionality comes from a few functions:

Key Components

Draw / Render (g) - Occurrences: 20
This is what renders most everything on the screen, through the use of the Graphics AWT.
It loads each component in the order it was first presented. Using the “Playing” Gamestate as an example, the background would load in before anything else, followed by the checks to ensure the game isn’t paused or over, and after passing all of them, continues to draw the remainder of the objects that make up a level!

Gamestate “Playing” Draw Function

```
/**
	* Draws everything that is intended to be visible while in a level/playing the game
	*
	* @param g - the Graphics where to draw the screen
	*/
	@Override
	public void draw(Graphics g) {
	// draw background first so everything else sits on it
	drawBackground(g);
	// if it is paused, only draw the background and the pauseOverlay.
	if (paused) {
		pauseOverlay.draw(g);
		// return so it doesn't draw anything else
		return;
	} else if (gameOver) {
		deathOverlay.draw(g);
		deathOverlay.update();
		} else {
		// if not paused, draw everything beneath this.
		levelManager.draw(g, xLevelOffset);
		hud.draw(g);
		enemyManager.draw(g, xLevelOffset);
		projManager.draw(g, xLevelOffset);
		player.renderPlayer(g);
		}
	}
```

ARCHITECTURE OVERVIEW continued

Run - Occurrences: 1
This is how we monitor the game's performance and ensure a smooth animation experience by forcing two variables to sync. After setting a strict amount of frames and updates per second, we set a timer to check between updates to ensure no frame is missing or lost out due to the game updating too fast.

```
/**
	* Handles the update aspects of the game, such as updates to logical processes and
frames
	* per second
	*/
	@Override
	public void run() {
	// How long each frame/update should take in nanoseconds
	double timePerFrame = NANOSECONDS_IN_SEC / FPS_SET;
	double timePerUpdate = NANOSECONDS_IN_SEC / UPS_SET;
	long previousTime = System.nanoTime();

	// this is the time that it has been the last update
	double deltaUpdates = 0;
	// this is the time that it has been the last frame
	double deltaFrames = 0;

	// while the game is running, so this should be an infinite loop
	while (true) {
		// get the current time in nano seconds
		long currentTime = System.nanoTime();
		// add the time since the last check to the delta's
		deltaUpdates += (currentTime - previousTime) / timePerUpdate;
		deltaFrames += (currentTime - previousTime) / timePerFrame;
		// this time is now the last checked time
		previousTime = currentTime;

		// if the delta update time is >= 1, then update the game
		if (deltaUpdates >= 1) {
		updateGameState();
		// only subtract 1 from the updates, so if it is above 1, that extra time is
accounted for
		// in the next update
		deltaUpdates--;
		}

		// if the delta frame time is >= 1, then repaint the game
		if (deltaFrames >= 1) {
		gamePanel.repaint();
		// only subtract 1 from the frame, so if it is above 1, that extra time is
accounted for
		// in the next frame
		deltaFrames--;
		}

	}
	}
```

ARCHITECTURE OVERVIEW continued

Update - Occurrences: 17
This is how we keep track of everything during the game loop.

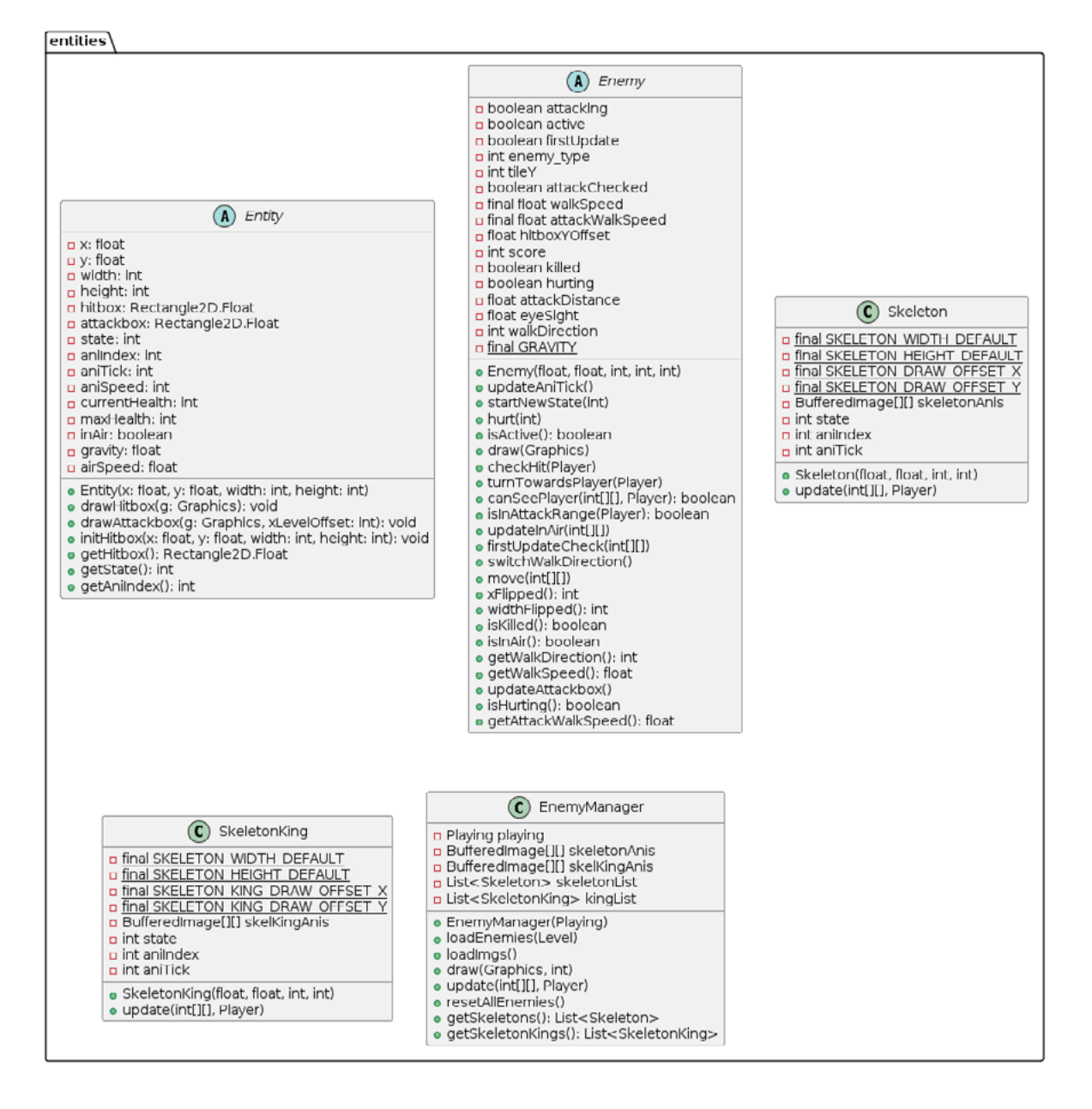
Player Entity's Update Function

```
/**
	* Handles updates for Position, Animation Tick, and Setting Animations
	*
	* @param xLevelOffset - how far the screen offset is from scrolling
	*/
	public void update(int xLevelOffset) {
	// if dead, only update animations
	if (killed) {
		updateAniTick();
		setAnimation();
		return;
	}

	this.xLevelOffset = xLevelOffset;
	updatePos();
	updateAniTick();
	if (attacking) {
		checkAttack();
	}
	setAnimation();
}
```

UMLS

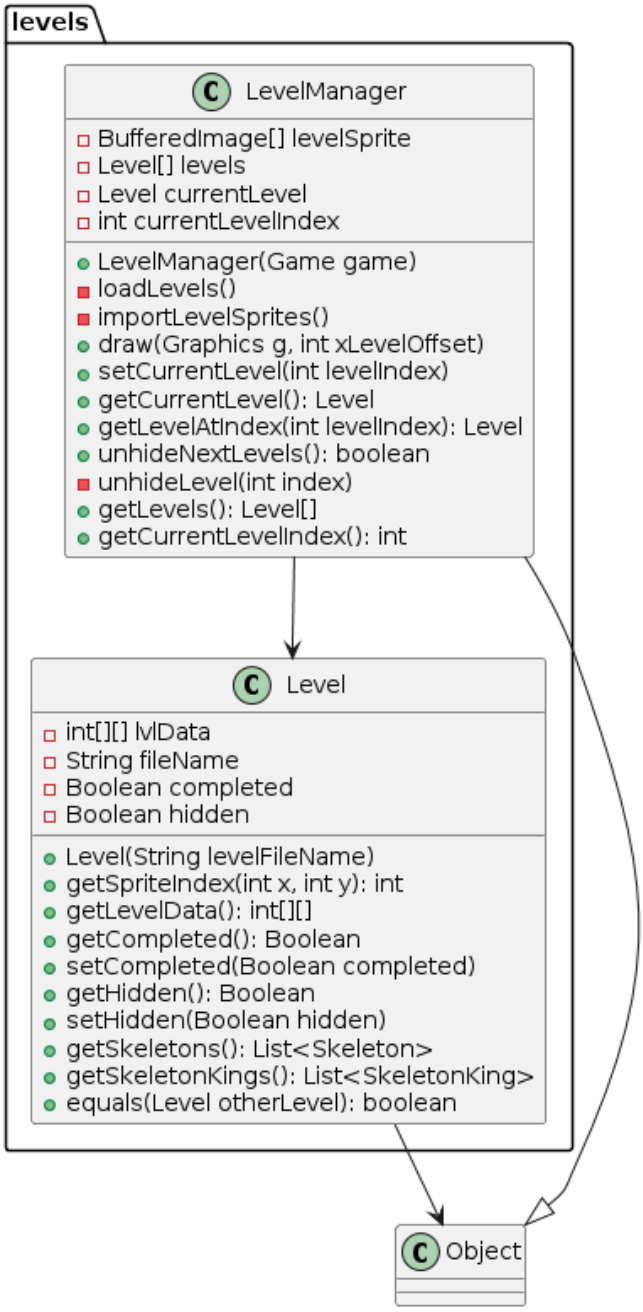
Entities



All Enemies are based off of the Enemy abstract class. The Enemy class is what handles most walking and fighting behavior, so that all Enemies have the same basic structure, the more specific behaviors are in individual classes. This allows for more, different, Enemies to be easily created. The EnemyManager class handles creation/ updating/drawing of all Enemies. It is cleared and refreshed each time a new level is loaded.

UMLS continued

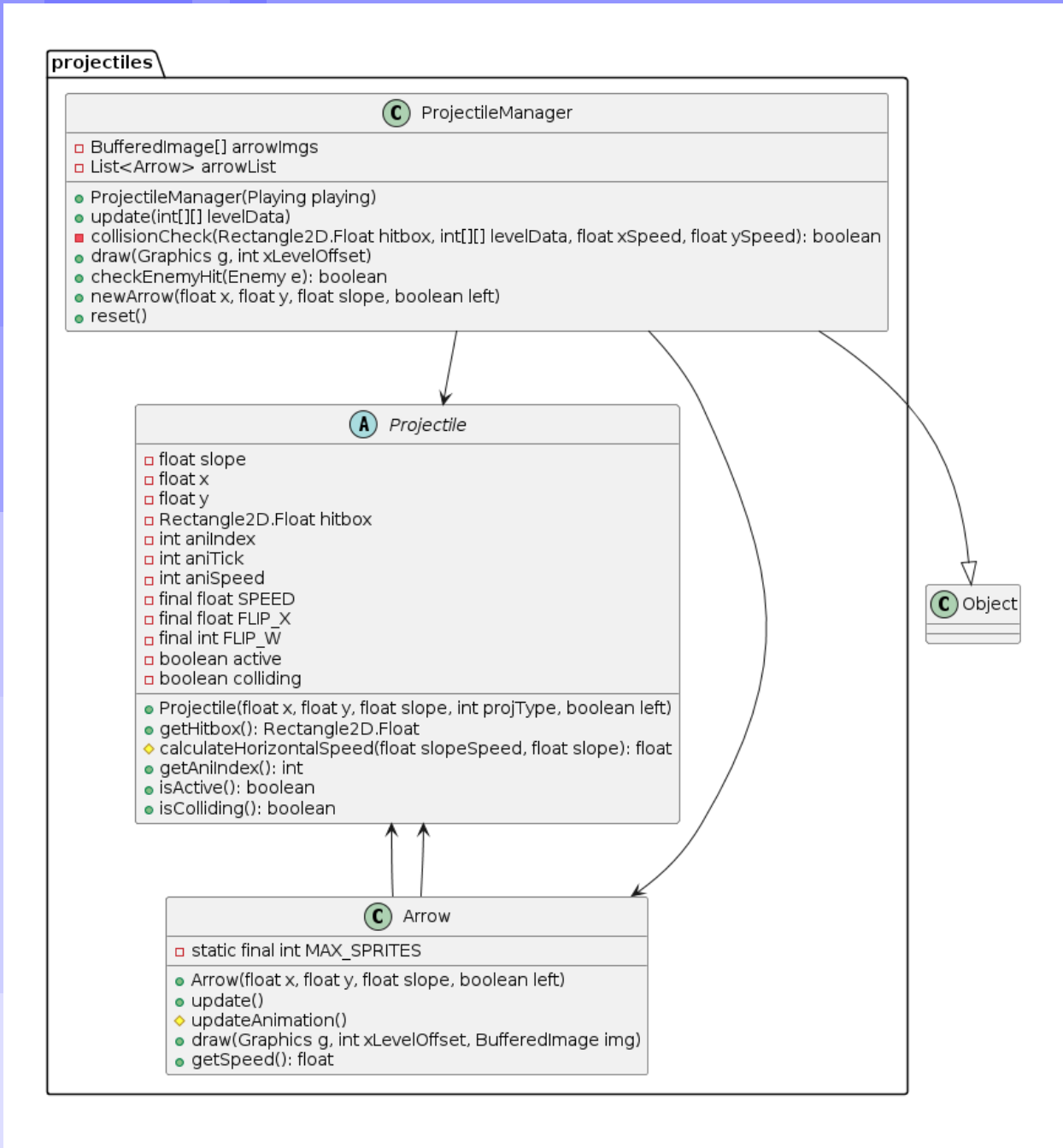
Levels



Every level keeps track of its state (hidden and completed). They also keep track of the 2D int array that represents the environment and how it should be built. The LevelManager keeps track of all levels and the current level being played. It is in charge of drawing the current level using the appropriate sprite sheets.

UMLS continued

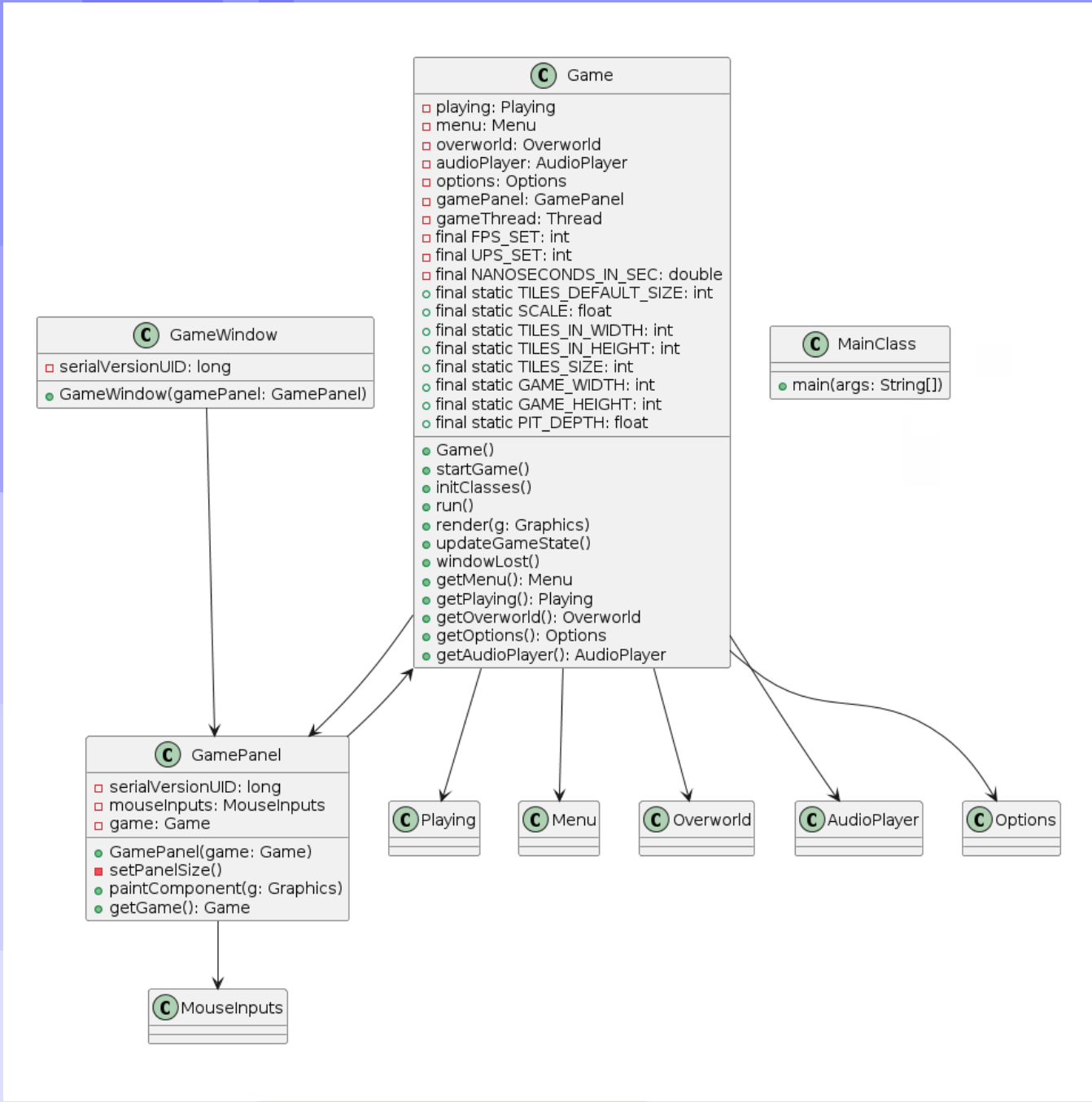
Projectiles



Similar to Enemies, Projectile is an abstract class that handles the general behavior of all flying objects. Arrow handles specific behavior to arrow. Other, new, game-objects should be based on Projectiles. Projectile Manager is in charge of drawing/updating/creating all Projectiles in a game.

UMLS continued

Main



The Game class handles all different GameStates (MENU, OVERWORLD, PLAYING, INSTRUCTIONS, and QUIT). It keeps track of how often to update/draw the current GameState and what sounds to play using its AudioPlayer. GamePanel and GameWindow are extensions of JPanel and JFrame, respectively, which are customized to fit the Game.

DEVELOPMENT PROCESS

Testing and Format

This is the format of how all tests were done. There is a main UnitTester class that tests all other drivers. Each driver tests a specific package. For example, the EntityDriver tests all concrete classes within the Entity package: Player; Skeleton; SkeletonKing; EnemyManager.

```java
/**
 * This will test all classes in the Entity package
 *
 * @author Sean-Paul Brown
 */
public class EntityDriver implements DriverInterface {

    /**
     * This method will be the main method of this class. It will test each class in the
     * entities package.
     */
    @Override
    public boolean test() {
        boolean allSuccess = true;
        // test the Enemies
        if (!testEnemy()) {
            allSuccess = false;
        }
        // test the enemy manager
        if (!testEnemyManager()) {
            allSuccess = false;
        }
        // test the Player
        if (!testPlayer()) {
            allSuccess = false;
        }

        return allSuccess;
    }
```

This also shows our format for how we structure our code. We followed javadoc standards, every class, method, and property had a tag. For our code code format, we followed standard formats. Additionally, we made sure that every block (if-else, for, try-catch, etc.) had curly braces surrounding it. Additionally each keyword is followed by a space and we used 4 spaces instead of tabs. Each class has an author tag that includes all of the authors that worked on it.

```java
boolean allPassed = true;

// test the entity classes
EntityDriver entityDriver = new EntityDriver();
if (!entityDriver.test()) {
    System.err.println("ENTITY DRIVER FAILED");
    allPassed = false;
} else {
    System.out.println("\tEntities passed!");
}

// test the input classes
InputsDriver inputsDriver = new InputsDriver();
if (!inputsDriver.test()) {
    System.err.println("INPUTS DRIVER FAILED");
    allPassed = false;
} else {
    System.out.println("Inputs passed!");
}
```

DEVELOPMENT PROCESS continued

Gantt Charts

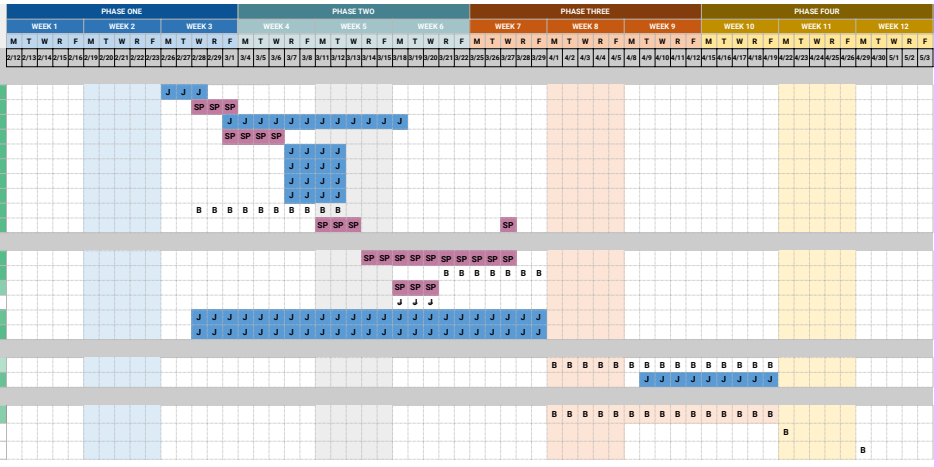
This is our estimated Gantt chart for this project. The “J” represents all of John Botonakis’ tasks. The “SP” represents all of Sean-Paul Brown’s tasks. It gives a very general idea of how all tasks (coding and non-coding) were divided and the pacing of the project. More details for the coding can be found in the javadoc for each class. The initial structure of this chart was made on February 12, 2024. As the project progressed, some small adjustments were made.

Game Gantt Chart

Smartsheet Tip ➜ A Gantt chart's visual timeline allows you to see details about each task as well as project dependencies.

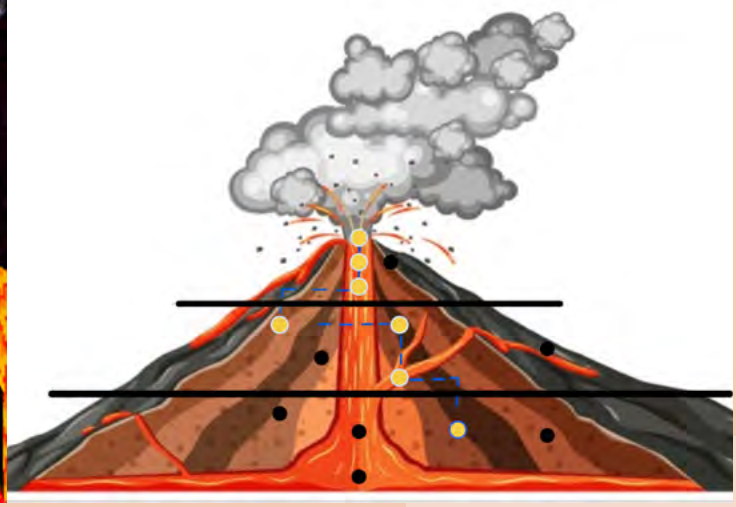
PROJECT TITLE	*ARTEMIS DASH*	COMPANY NAME	Dream Team
PROJECT MANAGER	John and Sean-Paul	DATE	2/14/24

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (days)	PCT OF TASK COMPLETE
1	**Project Conception and Initiation**					
2	**Game Engine**					
2.1	Drawing to the screen	John	2/20/24	2/28/24	8	100%
2.1.1	Keybinds / Control	Sean	2/28/24	3/1/24	3	100%
2.2	Collision Between Sprites	John	3/1/24	3/18/24	17	100%
2.3	Game States	Sean	3/1/24	3/6/24	5	100%
2.4	Movement Mechanics	John	3/7/24	3/13/24	6	100%
2.4.1	Jumping	John	3/7/24	3/13/24	6	100%
2.4.2	Dashing / Sliding	John	3/7/24	3/13/24	6	100%
2.4.3	Shooting with Mouse	John	3/7/24	3/13/24	6	100%
2.5	Objectives (aside from get to the end)	Both	2/28/24	3/12/24	14	100%
2.6	Static Enemies (No movement)	Sean	3/11/24	3/14/24	3	100%
3	**Gas in the Engine**					
3.1	Enemy Behavior	Sean	3/14/24	3/20/24	6	100%
3.2	Map Building	Both	3/21/24	3/29/24	8	100%
3.2.1	Obstacles	Sean	3/18/24	3/20/24	2	70%
~~3.2.2~~	~~Powerups [SCRAPPED]~~	~~John~~	~~3/18/24~~	~~3/20/24~~	~~2~~	~~0%~~
3.3	Art Assets	John	2/28/24	3/29/24	31	90%
3.3.1	Spritework	John	2/28/24	3/29/24	31	100%
4	**Manual and Finalization**					
4.1	Game Manual	Both	4/1/24	4/19/24	18	45%
4.2	Music + SFX	John	4/8/24	4/19/24	11	90%
5	**Wrap-Up**					
5.3	Polishing	Both	4/1/24	4/19/24	18	72%
5.1	Demonstration to Nate	Both	4/22/24	4/22/24	0	0%
5.2	Demonstration to Class	Both	4/29/24	4/29/24	0	0%



WISHLIST

Had we more time and more money **wink wink**, we would've fleshed out the world of our game a lot more. Things such as rudimentary cutscenes with text and sounds, additional worlds and levels, secrets to be found inside levels that would take the player to different areas of the specified world, and much much more.



In our original mockups, we had such features on prominent display, however we were unable to complete them due to the time constraint. Simple things such as Gems and collectables would be easy enough to implement, along with a better version of scaling the levels both vertically and horizontally, with the source code provided.

ACKNOWLEDGMENTS / THANK YOUS

Acknowledgments for contributors or libraries used in the project. All links work when clicked on in a browser.

Special thanks to KaarinGaming for his YouTube Tutorials and general project structure. This project would not have been as far along without those videos.

Visuals:
Lava Tileset - LudicArts - Link
Retro Pixel Ribbons, Banners and Frames 2 - bdragon1727 - Link
Free Dungeon Platformer Pixel Art Tileset - Link
Elementals Leaf Ranger - chierit - Link
Volcano Pack - samOki - Link
2 seamless lava tiles - LuminousDragonGames - Link

Sounds:
Free Fantasy SFX Pack - Tom Music - Link
COLORALPHA 50 Menu Interface SFX - Color Alpha - Link
5 sounds/short melodies - crazyduckgames - Link
Game Over - Jummit - Link
Generic 8-bit JRPG Soundtrack - AVGVSTA - Link
8-BIT Jumping Sounds - Jesús Lastra - Link
200 Free SFX - Kronbits - Link
Victory! - jkfite01 - Link

THANK YOU!

CONTACT INFORMATION

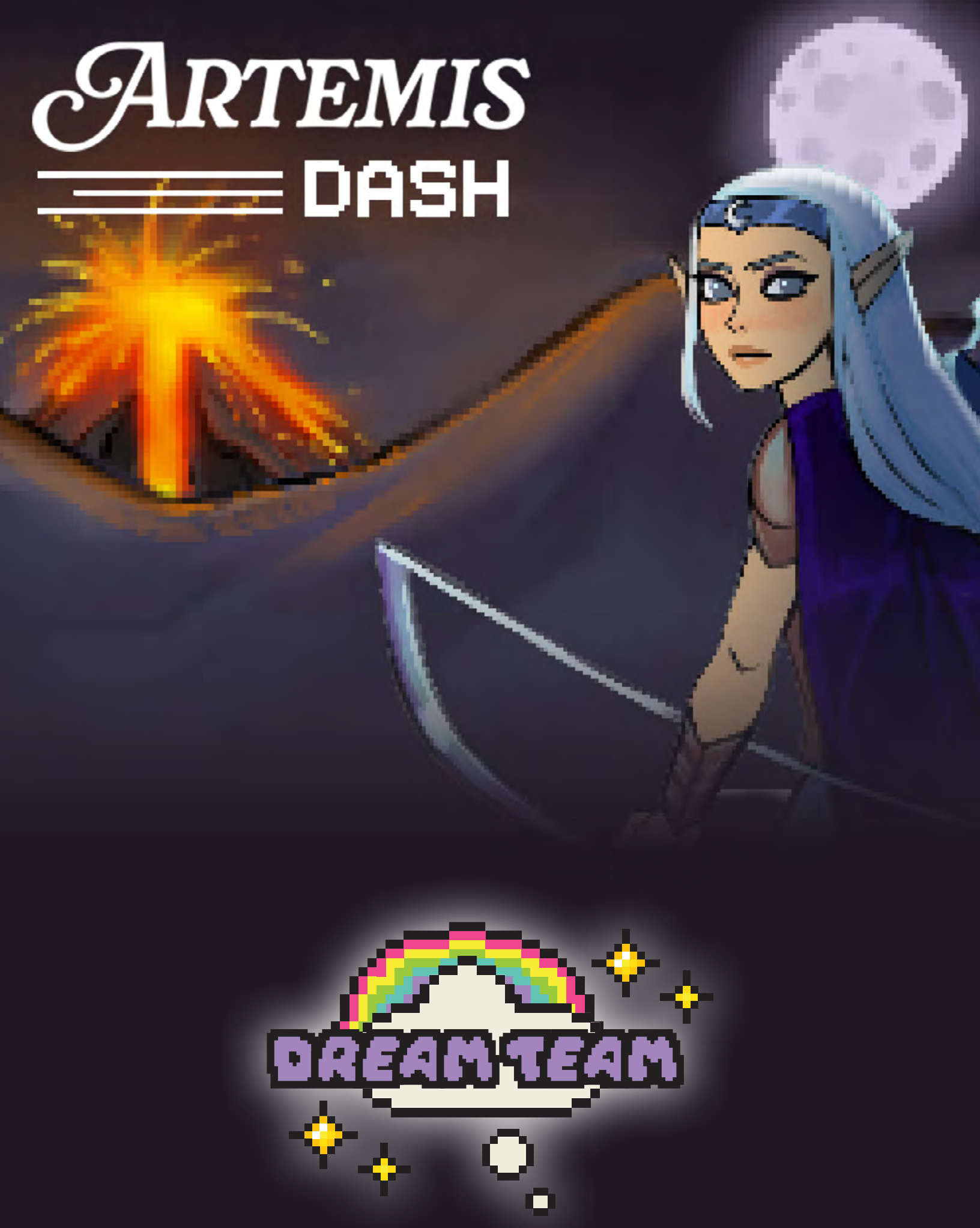
John Botonakis - Lead Programmer
Carroll College Computer Science Major

Sean-Paul Brown - Lead Programmer
Carroll College Data Science Major

Lindsey C Sewell - Art Direction
BFA in Media Arts

QR CODE TO GITHUB / JAVADOC:

ARTEMIS DASH



DREAM TEAM