RELATIVITY

Produced by the **Gucci Zeng** team:

Jacob Frank
Eddie Tang
Jerry Zeng

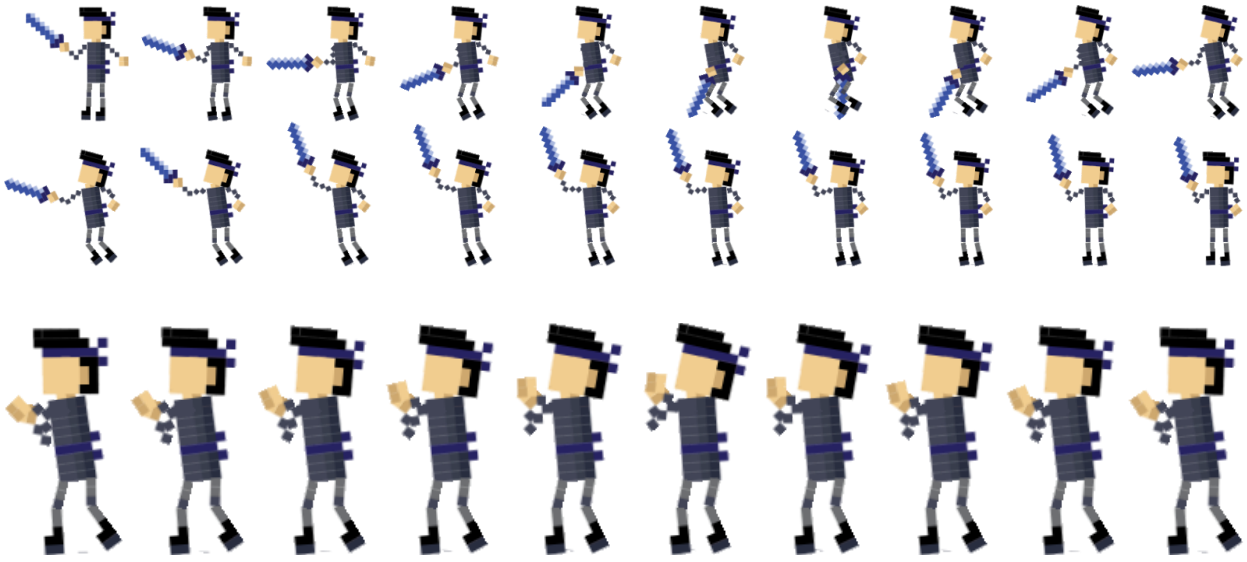
Abstract:

Relativity is a bullet-hell platformer. Time only moves when you move.

Art & Music Direction

Graphics: The game has a vector and flat minimalistic artstyle. The design has a high emphasis on contrasting colors, in order to increase playability and decrease screen clutter. The art style is meant to increase user attention on the action happening on the foreground. Enemy objects are mostly themed red, and the player's objects are colored blue, while mostly everything else is washed out white/grey. Platforms with special abilities may be colored differently(for example, platforms you can jump higher off of may be colored green, while platforms that give you a speed boost may be colored yellow.)

Concept art of the playable character (Currently pixelated for simple animation and importing):



UI Design: The UI itself is composed of a main menu, a pause screen, and a game over screen. The UI is going to follow a similar graphical design set in the previous paragraph. Includes a flat minimalist art style composed of vector-style art, with contrasting colors and readable fonts to maximize ease of use.

Soundtrack: The soundtrack is based off of the EDM and house genre, with inspiration taken from movies such as Heat and Outrun and video games such as Payday and Hotline Miami. It features a heavy use of synthesizers and 808s, setting an intense and energetic mood for the player.

Prototype Sample Title Screen 1 Music - https://drive.google.com/file/d/1p_3ow9m1xYH9EDTJi-K5fljC4gkdZh0k/view?usp=sharing

Sound Design: There is an emphasis on the sound of projectile motion along with time dilation. Slow-motion sound effects play when the player is in a state of slow-motion, and normal sound effects play when the player is out of slow motion. The lack of superficial sound effects is meant to focus the players attention on the sound of enemy projectiles, providing a warning of incoming danger.

Game Mechanics

Time depends on your movement speed, the slower you move, the slower time moves around you, and vice versa. You are given the task of maneuvering through a level of platforms and turrets in order to get to the end. To help you with this task, your playable character is given the ability to manipulate time depending on his movement speed. The main obstacles placed between you and victory is the numerous traps and turrets which you must either avoid or get past successfully. For example, a player may be required to jump over a spike pit, or dash past a piston, while dodging a number of bullets and lasers. By limiting the player to only being able to dodge these obstacles instead of destroying them, the game utilizes a players reaction time and critical thinking, providing a unique experience in the platformer genre. Along with the obstacles present, the level also contains a number of special platforms. These platforms can provide a number of buffs to the character, ranging from speed increases to an increased jump height. In order to maximize the replayability of the game, the player is also timed on each level, with a higher score obtained the faster a player completes it. This plays into the time manipulation mechanic, as this slowdown/speedup mechanic not only allows new players to become familiar with the game, but allows experienced players to run through the game and a faster pace. With a leaderboard to keep track of every run the player accomplishes, the player is encouraged to further learn and master each level beyond their first completion. With these simple but innovative mechanics, we believe that *Relativity* will provide limitless replayability and prove to be fun for everyone.

Workflow

Assets are created while the code is being developed. The code uses placeholder images and audio until the assets are produced and then imported into the game. Assets are created whenever necessary; (ie. a new enemy concept), and as a result, code will start being produced for that concept at that time as well. The initial focus is creating a coding environment that will allow us to easily produce levels. After we create the initial mechanics, we will focus on making levels that will feel good to beginners, as well as older players, who want to brawl through this game. After creating levels we will add the more aesthetic details that will make the playability feel smoother and give more feedback, such as particle effects and a HUD that doesn't demand too much attention from the player.

Team Roles

Jerry:

Lead Graphic Designer - Does the art and animation of the game's assets
Head of Design - Influences level design and the art/audio direction of the game to achieve
Programmer - Concepts, prototypes, and writes out the code needed to run the application, along with bug fixing and error handling.

Jacob:

Lead Programmer - Concepts, prototypes, and writes out the code needed to run the application, along with bug fixing and error handling.
Level Designer - Makes sure platforms and enemy placement are in locations where the game feels challenging but still feels fair to play.

Edward:

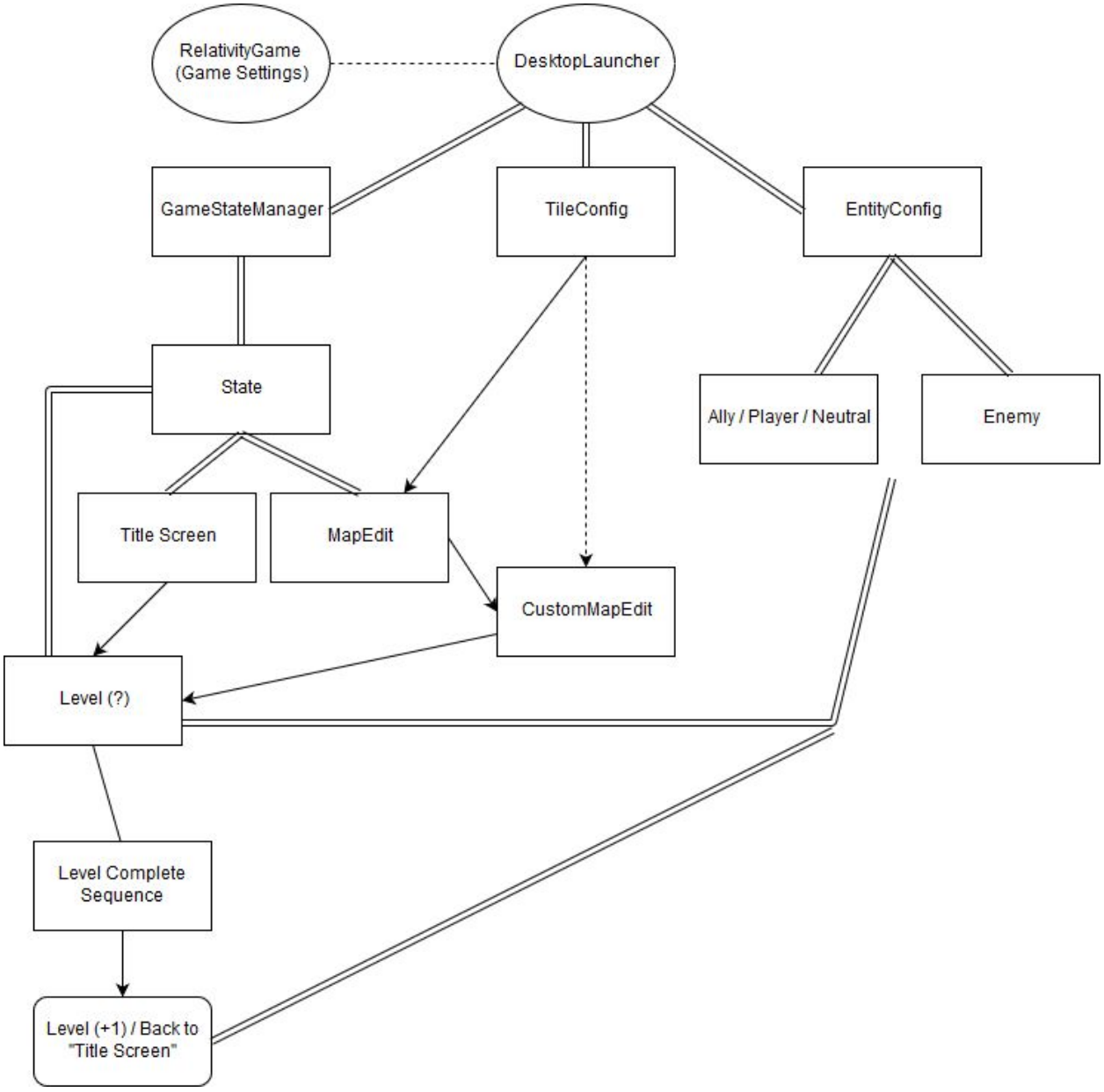
Creative Director - Organizes ideas, concepts, and the direction of the video game
Lead Audio Technician - Responsible for producing and mastering the soundtrack, along with producing the sound effects.
Tester - Tests the product for bugs and reports them to the lead programmer.

Code Details and Structure

LibGDX – LibGDX is an application framework written in the Java programming language with some C and C++ components for performance dependent code. It is the basis of our project's code, and complements our existing ability to work with java.

Diagram – Gives a general overview of the structure of the code by class name and placeholder names. Bubbles nearing the bottom are subject to further configuration in the future. A save feature, as well as other features are currently being developed.

DesktopLauncher	This is the class with the main method. This class declares important constants such as the system's width, height, the FPS, and launches the game.
RelativityGame	This is where the game settings variables are stored as well as where the game is first rendered at. All save-game data and custom level information will be stored in the documents folder of the host PC.
GameStateManager	Manages every state of the game, whether it be a pause screen, level sequence, or main menu.
TileConfig	Stores information on the configuration of the tiles and how to load them from the map publisher/editor.
EntityConfig	Stores a basis for all entities in the game to follow. Will be configured to hold similar characteristics of all entities in the game, including the player's.
State	Holds the basis for all states the game can hold. This is essentially an addition to the GameStateManager, but provides a template for all states, such as the title screen, to follow.
Title Screen	The main menu of the game. Will have an option to view credits, design a level, and start a level based on a saved game
MapEdit	Will provide the functionality and UI to create a level
Level (?)	Holds level information and compiles map save data to create the level corresponding to the (?) placeholder. Grabs entity information necessary to build the level.
CustomMapEdit	The configuration of how the MapEdit state performs. Will hold specific configurations vital to the operation of the MapEdit class.
Level Sequence Complete	A state that occurs after the title screen. Will branch off with 2 options to return to the title screen or continue onwards with any further levels.


RelativityGame
(Game Settings)
DesktopLauncher
GameStateManager
TileConfig
EntityConfig
State
Ally / Player / Neutral
Enemy
Title Screen
MapEdit
CustomMapEdit
Level (?)
Level Complete
Sequence
Level (+1) / Back to
"Title Screen"