Can't Escape Games

GAME DESIGN DOCUMENT

DigiPen's Team Can't Escape Games

Team Members:
Albert Harley
Aleksey Perfileve
Arnold George
Jose Rosenbluth
Ramzi Mourtada

Gam 550
DigiPen Institute of Technology

December 9, 2019

Contents

NOTE: When testing the levels, you can enter or exit the levels using the Escape Key or the Select Button on the Joystick. Keybinds for each game can be found in the respective prototype controller description.

Prototypes 1 and 3 can be played with keyboard or joystick.

Prototype 2 can only be played with the mouse as an FPS Controller.

Prototype 1: 3rd Person Linear Adventure

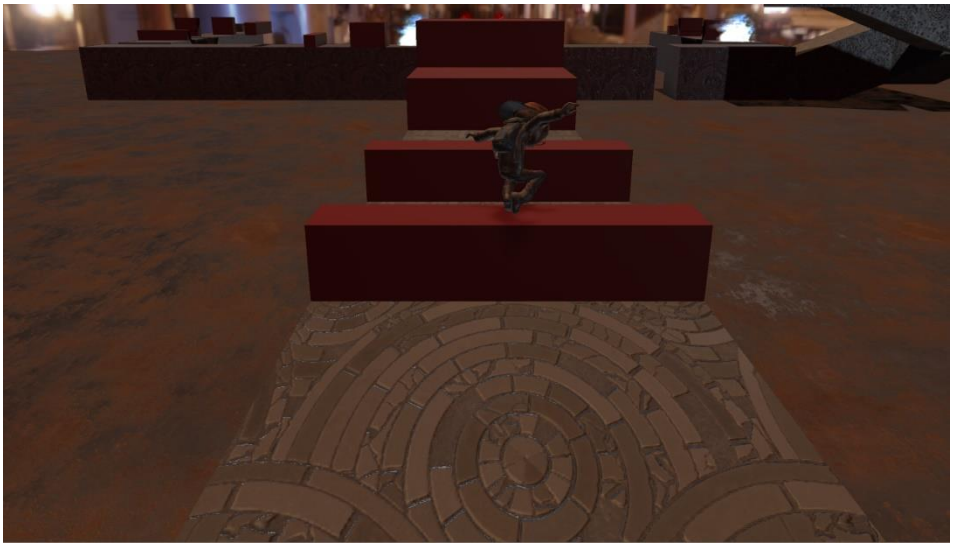
The purpose of this prototype was to demonstrate our engine's versatility in building a 3rd person camera adventure game that is linear in nature, but also includes some going back and forth before being able to reach the final objective.

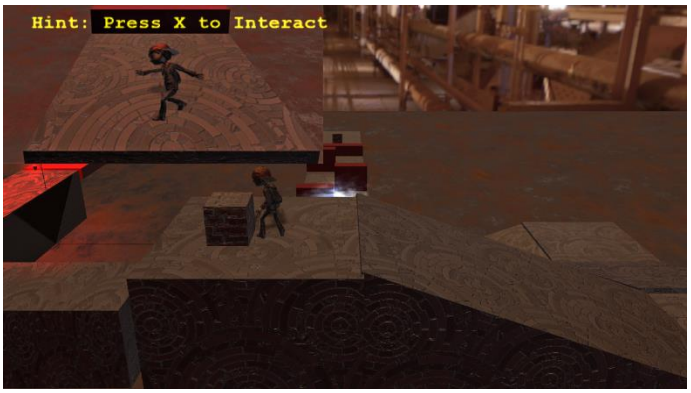
Some levels can be built to be completely linear and straightforward, while others can utilize the mechanic of having to run backwards (either to solve a puzzle or to run away from a certain hazard).

In this document, we will describe the different gameplay mechanics used, camera usage, and the gameplay testing that we performed to get the provided level's design.

Player Controller

Move: Using the Joystick's **left stick** or the Keyboard **WASD** buttons.
Jump: Using the Joystick **A** button or the Keyboard **Space** button.
Prone: Using the Joystick **B** button or the Keyboard **B** button.
Interact: Using the Joystick **X** button or the Keyboard **X** button.





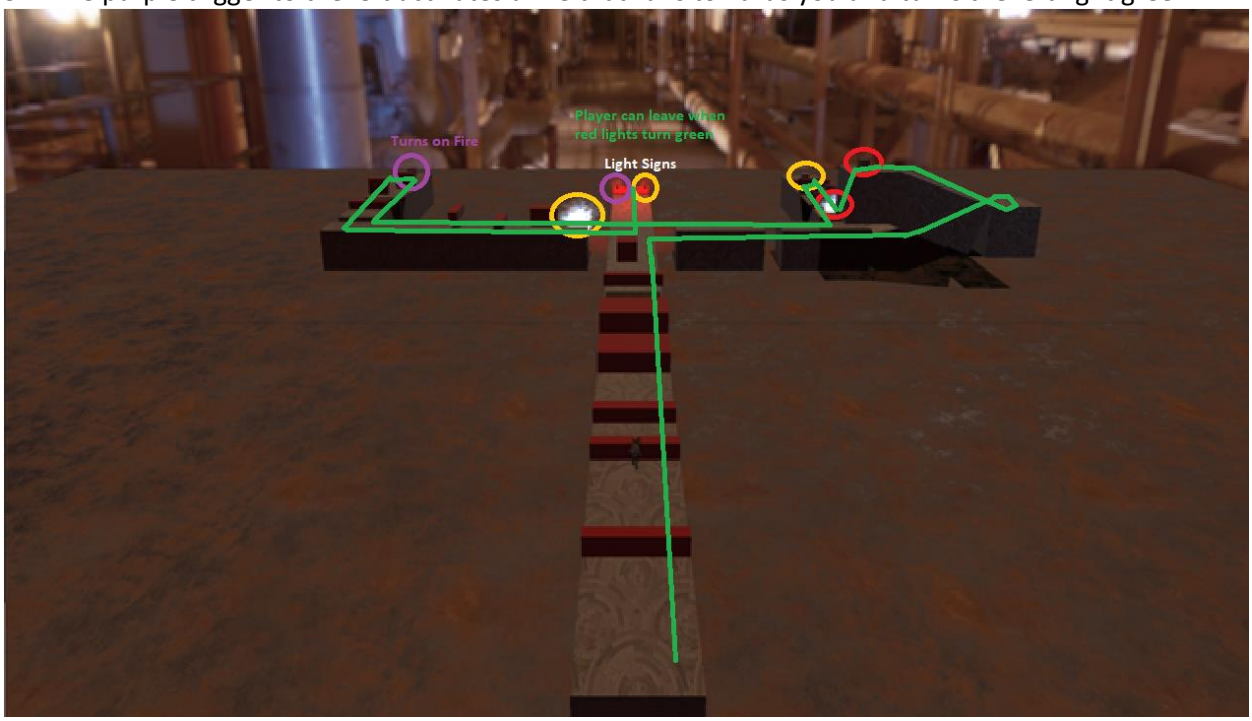
Game Mechanics:

In the level we demonstrated we used two lights at the end of the level to signal whether you are clear to proceed or not. At the start, the two lights are red. After solving the puzzles in the level certain triggers will turn their respective lights green. When both lights turn green, the player can proceed to the next level.

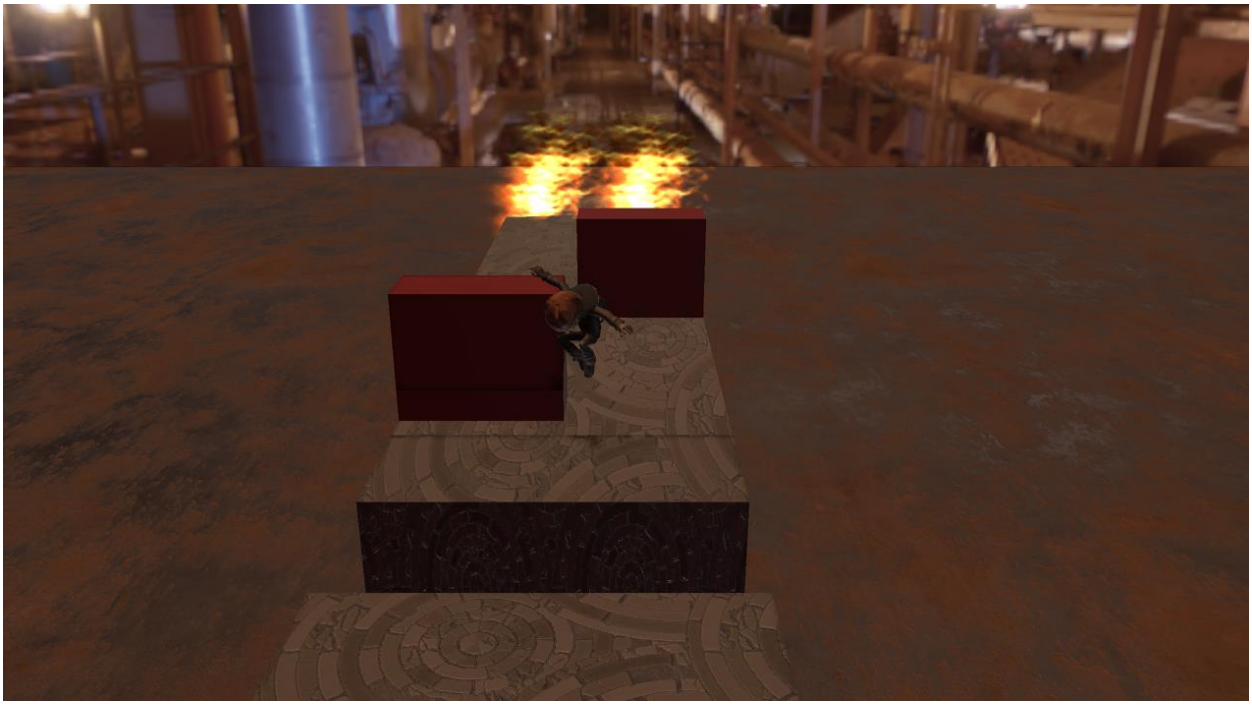
The figure below shows the player's path in order to finish the level. This is the case where we utilize having the player move in all axes. The main gameplay focus is being able to run in $\pm$z and $\pm$x axes, while the camera is maintained in the -z axis. We also showcase the ability to use physics with the player controller to perform crouching under certain obstacles or going up certain planes.

Objects with circles of the same color indicate they are bound to the same trigger.

1- The red trigger on the pillar top deactivates the first blue firewall below it
2- The orange trigger to the right deactivates the second blue firewall and turns the right light green
3- The purple trigger to the left activates a fire that runs towards you and turns the left light green



The last part in the game includes the adrenaline raising part where the game is now clearly doing things against you. Previously, interactive triggers would either turn a fire off, or a light green. In the last part, the trigger turns the light green, but also activates a fire that is out to catch you and kill you. The player now must run backwards and try to anticipate all the obstacles on their way back so that they wouldn't get caught by the fire. This part is meant to be the peak of the interest curve before bringing it back down and having the player run back to the center of the level where the elevator is now active.



Gameplay Testing and Camera Decisions

The aim of this playtest is to discern 3 main factors in gameplay:

1. Identify engagement of different types of obstacles the player must interact with:
    a) Gaps to jump over
    b) Obstacles to jump over
    c) Obstacles to prone under
    d) Obstacles that move in a pattern and push you off the ground
    e) Obstacles that are deadly when they collide with you
2. Based on the different environments identify a camera style that can adapt to all kinds of objects and interactions
3. Define what objects that can or cannot be used when running forward, backward, left or right.

For the purpose of this playtest, the subject is Francesco Morando, a sophomore BAGD student at DigiPen. Due to his major, we believe he could provide us with useful feedback on game design regarding the game's mechanics. On the other hand, since he's a BAGD, he might point out and pick stuff that are not extremely relevant to the development stage. Once the playtest is done, we will skim through the feedback and discern the most important notes from the less relevant ones. Accordingly, we can move on to the Beta stage with a clear vision of the design objectives.

Script:

The level that Francesco will be playing consists of two main parts. The first includes a part where he has to run the actor in the forward direction (similar to the first part of the level we demoed), but must run back once they reach the end. This part consists of small platforms the player must ***jump over***, and higher platforms that the player must ***jump on***, a **moving obstacle** he must run near without falling, and an obstacle he must ***prone under***.

The second includes the same types of interactions but in the horizontal directions, with the addition of an inclined plane that the player must run horizontally upwards.

The main purpose is to find the optimal position for the camera, and whether it needs to change depending on the direction the player is moving in. Since this game is linear in its nature, the main challenge is to be able to build levels that are fun to explore, challenging at the same time, and not have any limitations on variations around the game's interest curve.

Playtest:

Obstacle Concerns:

All the obstacles seemed fitting for this kind of game, and the playtester had fun dealing with all of them. The only issue was that it may be a bit overwhelming when the player must run backwards with moving obstacles in the environment they are running into. This was problematic as sometimes he would realize that he is near a moving object by the time it is too late to stop, and the object is already pushing him off the platform. Using this kind of mechanic seemed a bit difficult and may only be useful until later parts of the game where the player has some higher expertise in the game. Moving obstacles that run behind the player (such as the fire at the end) when they are running backwards however seemed like a good combination in making those parts more exciting.

Camera Concerns:

When starting this part, we used a camera set at -10 degrees, set at 5 units behind the player and 1 unit above them. The field of view was adjusted before the playtest to allow proper vision for the platforms at 50 degrees. The camera was set to follow the player as they move throughout the level.

The issues became clear from the first few moments of the playtest.

The first case was after falling from high obstacles the player musts stand on. When falling and having the camera follow the player downwards, it goes inside that obstacles. This is very annoying as the player cannot see where they are landing as their vision is blocked in integral moments (landing) by the obstacle.

The same issue became amplified when running backwards. Any obstacle that is high that the player must prone under or jump above becomes a vision blocker on the way back. It seemed clear the comments would be around this issue, and a more interactive camera would be more useful.

In the horizontal part of the level, the issues were less obtrusive as no obstacles would stand in the way of the player. The main issue in this part was that a fixed camera in left and right was a little bit dull and creating room where the player can move without the camera constantly following would be better. When running upwards as well, this would seem better.

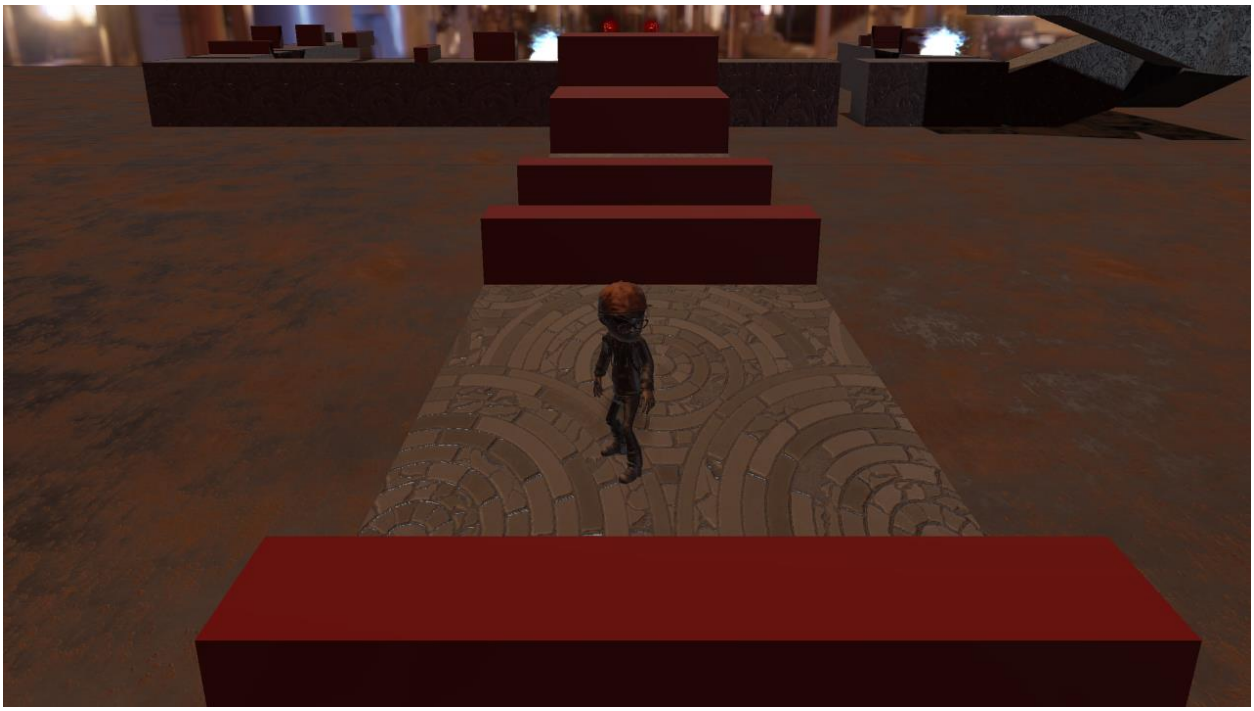
Results:

In order to fix the camera vision issues with obstacles, we needed to figure out a way to keep the camera system simple without having triggers to interpolate the camera every time we have a large obstacle. With some basic Geometry and some trial and error, we found an optimal position in which the camera and the environment must be given the following set of rules:

1- Set the camera at -20 degrees downwards
2- Have the camera follow a box around the player
    a. X: [-2,2] units => Symmetric in left and right. Edge of camera is around 5 units left and right which means player sees one quarter of screen outside box
    b. Y: [0, 1] units => Whenever the player falls, camera falls. Not the same going up.
    c. Z: [0, 2] units => If player moves back, camera goes back. Not the same moving forward.
3- Move the camera backwards slightly and upwards when running backwards. We used a linear interpolation along the line Y=-0.5z which follows the -20 degrees setup
4- Make sure all objects are never 2.5 units high (with respect to the ground the player is on). This ensures that no matter what the obstacle is, even if you can't see the player because they are behind it, the camera would still move above it. This can be later integrated in a game by having gaps in the environment that "coincidentally" allow the camera through. In this case, it is important to note that the camera blocking the player is not as annoying as it is when described previously because it is not sudden, and the player anticipates it.

Camera Example: Near

Camera Example: Far



Prototype 2: 1st Person Rail Shooter:

The main purpose of this level was to prototype a simple form of a 1st person camera game and measure the difficulty of creating a good player controller for such game types within our engine. Accordingly, we went with a rail shooter style game, inspired by the likes of House of the Dead. We believe this was a successful prototype in terms of proving the ability of our engine for creating a camera and a level that fits this genre.

Player Controller:

In this level, you must press the **Z button** to start the game and place the player on the rail. Once the game begins, you can use the **mouse** to aim and **left click** to shoot at the enemies.

The game ends when your health bar goes to 0.

Game Mechanics:

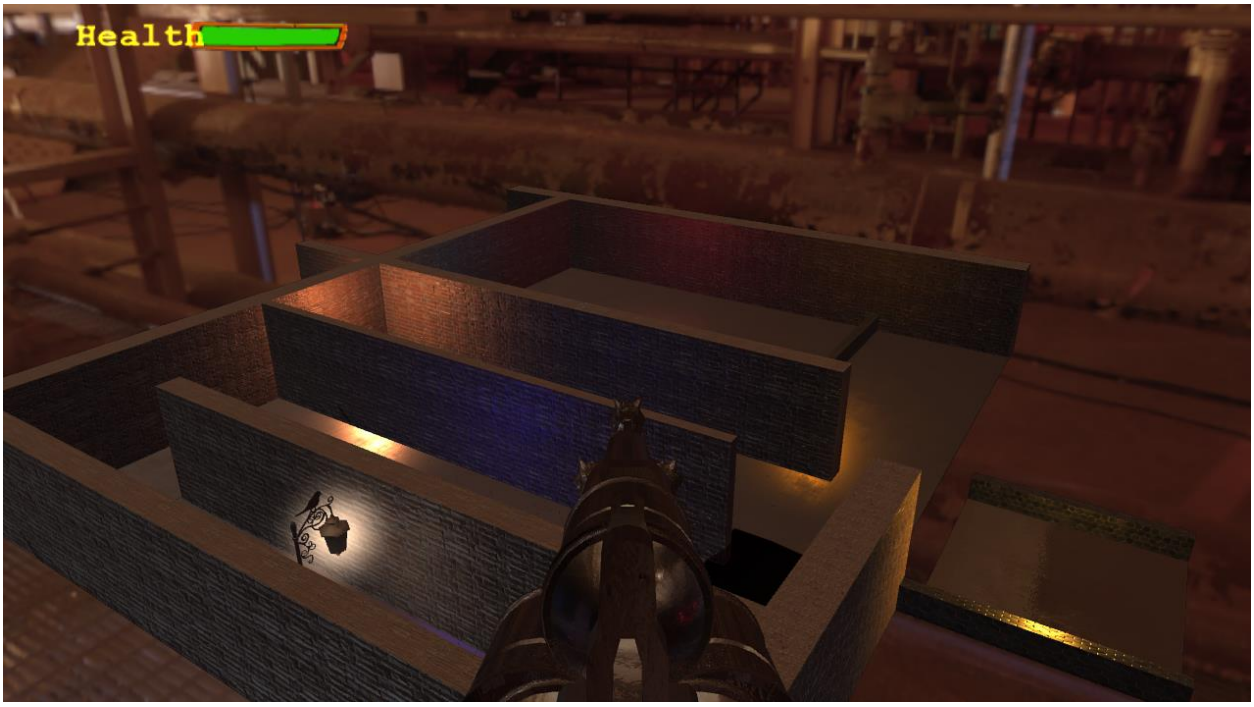
Here are some images of the player fighting with enemies moving on a curved path as they attack him. The focus in this game would be creating different curves and AI scripts that are hard for the player to anticipate or predict and use that to increase the engagement level throughout the game.

Challenges:

The main challenges in building this game where the following:

1- Design a level with corners and cover for player and enemy movement
2- Providing an easy scripting mechanism for using spline curves that the player and enemy AI can follow
3- Creating a player mouse controller that reflects a Doom Style shooter
4- Using different texturing and lighting to create a closed environment for the player.

Below is the result of the level we built for the given game genre. The focus is on having tight corners that can surprise the player as enemies come from different angles.



Even though it was interesting to build an FPS level, and it may be fun to play this for a while. The main problem with rail shooters is that it feels like you are stuck as the game moves you around. Being able to control the aim only is not free enough and becomes boring after a few minutes of gameplay. We did utilize our curve generation tech for moving the camera, and this technology can be used in any game and not just for a rail shooter.

Prototype 3: 3rd Person Infinite Runner

For this level, we wanted to experiment with the idea of procedural generation on the engine. How easy would it be to generate content as the player advances through the stage? and, how can we make this a fun concept? To answer this last question, and since the prototype in itself is a simple one, we added a self-competition element to it: The main goal of this prototype is to try and make it as far as possible and, as a bonus, gather as many coins as you can. When you die, the game displays how far you made it, and then the player can retry to surpass their previous record. In the next section, we will go through the main mechanics and elements of this prototype.

The whole runner consists on a infinite stretch of lava, with a few blocks to stand in between and **obstacles** spread along the way which you need to avoid, as they hurt the player. To go from one block of ground to the next, the player needs to make use of the **jetpack** mechanic. But you cannot fly forever, as there is a gasoline bar that gets depleted as the jetpack is used, and the player will crash into the floor/lava when this meter hits zero. In order to avoid this, a handful of gasoline tanks are distributed in semi random locations in between ground blocks, and the player has to fly smart in order to not run out of gas in the middle of the lava passage. As the level progress, the amount of distance you have to fly becomes larger, and the number of obstacles increases. So, the main idea is to be careful with the gas spending, position yourself strategically to grab the tanks while avoiding obstacles, and make it to the next safe ground.

Player Controller

Running: This is done by using the **left joystick (or Keyboard direction buttons)** while in the ground. The player will just run and turn in the direction he is walking.

Jumping: This is achieved by pressing **A (or keyboard space)** while grounded. The player will be given a small impulse up, and while in the air, it can still be controlled in which direction it moves.

Jetpack: When holding **A (or keyboard space)** while jumping or while falling, the player will activate the jetpack. This will give a constant propelling up, which allows the player to fly through the level. While flying, the directional stick can be used to move the player around as if he was grounded. This move uses gasoline.

In-air Dash: This move can only be used when the player is on the air (either jumping, falling, or flying). To activate, press **X** or **Y (or keyboard B)** on the controller. It will give a boost in whatever direction the player is facing, and it's a good move to use when trying to reach an item before it leaves the screen, or for fast avoiding obstacles (or reaching a platform faster). Warning though, this move uses a big amount of gasoline, so only use it when sure you can replenish.

The choice to give the player option to move forward and back was to have the freedom for the player to do stuff like going after items that are ahead and then, if there is enough gas, come back to pick stuff that may have gone past. In short, to add a tactical element to the player's choices.

Game Mechanics

The player, apart from having a gasoline bar, has a health bar. **Three** hits against obstacles will end the game, and there is no way to heal during a run. When getting hit by an obstacle, the player will gain a one second invulnerability. After that, it's back to being careful.

Lava: Number one hazard in the game. Touching the lava means instant game-over, no matter how much HP the player has left. So, avoid it at all cost.

Ground Walls: Clashing against the walls of the blocks of ground will hurt the player, taking 1 HP from them and knocking them back.

Obstacles: These will be spawned on semi random locations throughout the run, and as the distances between ground blocks increases, so does the number of obstacles (and the scrolling speed of the level increases too). The obstacles rotate at a semi-random speed, which makes it harder to avoid them, and will always be positioned on top of the lava, never on the area of the ground blocks. They will also take 1HP off the player and have the same knockback as the walls.

Ground blocks: These are the safe spots for the player. In here, you can stand and recover some gasoline (the player will recover gasoline not only by getting the tanks, but also by staying grounded, though at a slower rate).

Gasoline Tanks: These will recover 20% of your gasoline tank. They are randomly generated along the level, and there are always enough for the player to be able to remain on the air. But the obstacle placement can sometimes make it hazardous to go after some, so the player needs to choose carefully which to pick up.

Coins: Coins are just a collectible, with the amount collected being displayed on the upper left side of the screen. In the original planning of this prototype, coins were to be used to obtain instant powerups while playing, which could be chosen from a list in the upper right corner of the screen. These would've cost several coins and would've lasted a fixed amount of time after activating them. Because of time constraint, this feature did not make it into the final prototype.

Level Structure

As stated before, the level will consist on a big lava hallway, with small blocks of ground in between. As the player progresses, the scrolling of the level becomes faster, the amount of distance between ground blocks increases, and both the coins, gasoline tanks and obstacles increase in number. This is all handled procedurally.

Obstacles, Coins and Gasoline Tank Spawn

These resources never spawn on the area of the safe ground. They will always be positioned on top of the lava sectors, to incentivize the player to focus on recovering when on the safe ground.

Ground area on the level

Game Over

Player loses the game when HP runs to zero, or when hitting the lava. This will pop a text saying the game is over, and also showing the player how much progress they managed to achieve.

Conclusion:

Our conclusion from the given prototypes that if we are to build a game out of the given prototypes, the 1st seems to be the most scalable, while the 3rd has some very fun elements that could be added into the 1st prototype.

Although the 2nd prototype may not have enough elements to build a unique experience with an engaging interest curve, it does showcase our engine's ability for having smooth curve-based camera guided movement that may prove also useful to our 1st game prototype.