In the project folder you will find the newest build of the game. The game is made in Unity 2018.2.6f. As this is an older version of Unity I have made this document to describe some of the assets more in detail. I will also direct to specific locations for scripts, so you can open them without opening the project.

About this project

RoboRun was developed during my internship semester at DADIU. During the internship I created a game as a part of my "competence project".

Before the project I had wanted to create an infinite runner for a while, so I took the chance and made it.

My roles

As this was an individual project, I took the following roles:

- Game designer
- Programmer
- QA

Assets and animations were found on Unity Asset Store and Mixamo.com

The story

A war is about to break out and the military is creating destructive robots. You're the fourth in line, but you do not want to hurt anyone – Therefore, you run!

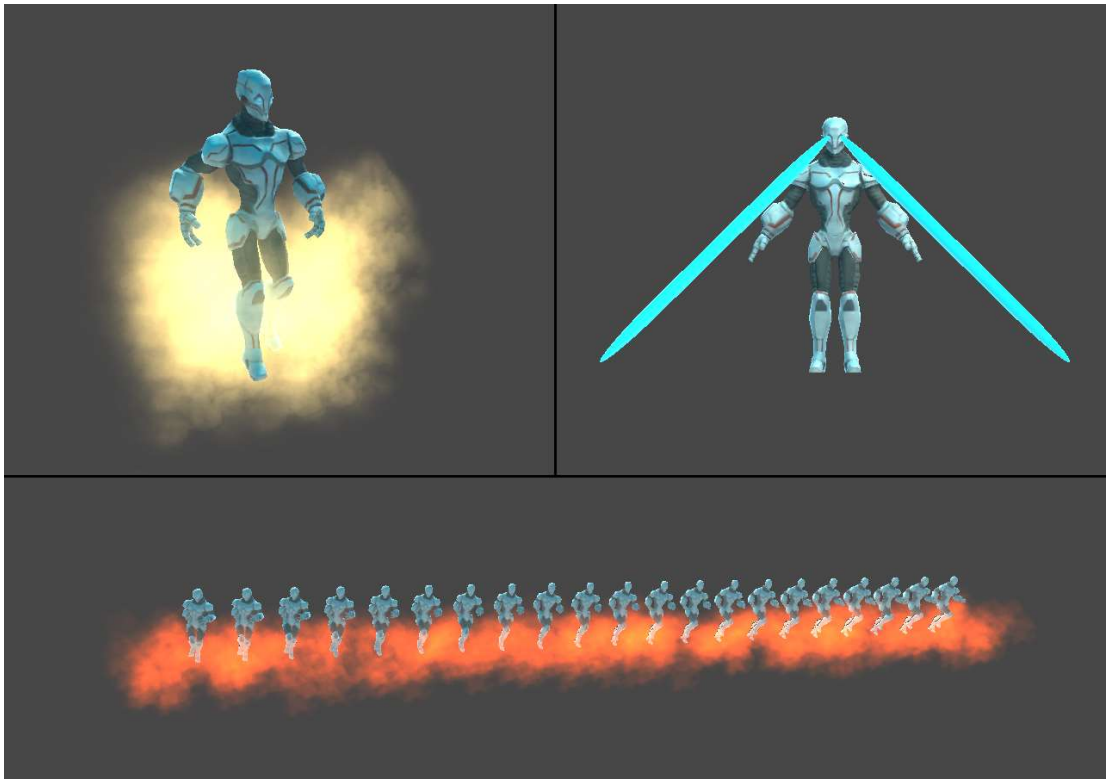
As you run, you must avoid the previously created destruction robots, hoping to find an escape. However, there is no end and you will just have to see how far you can get.

You always move forward. The left mouse button will turn you left; right mouse button will turn you right. By holding both buttons, you will walk. A screenshot from the game can be seen in the figure below.

Mechanics

Enemies

I wanted the challenges to consist of few enemies, which could be moved to create various challenges. The enemies can be seen in the figure below.



The chaser will walk toward you, so stay on a distance. The chaser will try to move in front of you, to block your path.

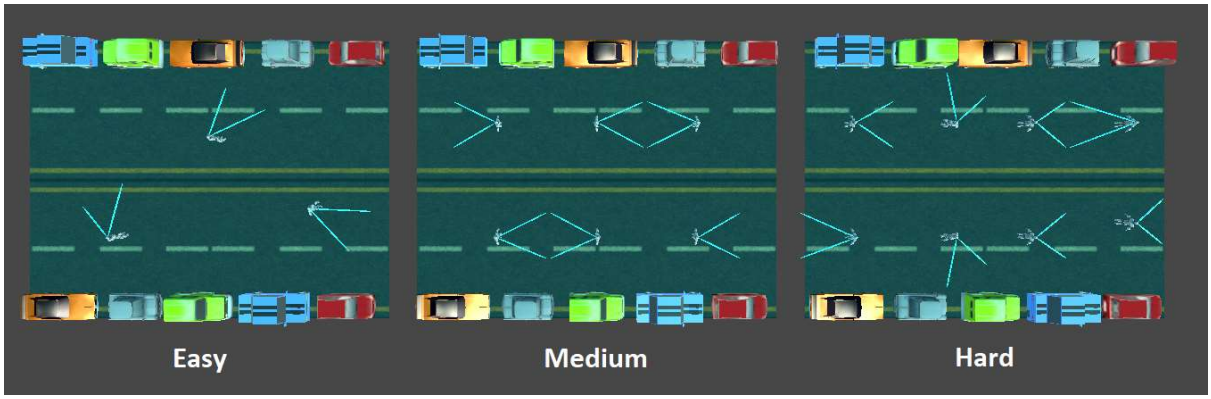
The sentry shoots laser out their eyes. The sentries will constantly turn with a speed determined by the difficulty.

The wall will not stop until they reach their target – you. This enemy is only in the game to keep the player running. The game would be too easy, if the player could constantly walk without any consequences. A rubber band effect was implemented, which would ensure that the wall runs slower as they get closer to the player.

Challenges

During the game, the difficulty increases. There are five different challenge layouts, which each has an easy, medium and hard version. The figure below shows how sentries are placed according to the same layout between difficulties.

The game starts with three of the easy layouts and slowly replaces them with new ones as the game progress.



Code

Challenge swapping

The script referenced in this section can be found in
"*RoboRun\CompetenceProject\Assets\Scripts\ChallengeSpawnerr.cs*

When you start the game there are two pools to determine which challenges to spawn: one for *current challenge candidates* and *future challenge candidates*.

Three of the easy layouts are picked at random from future candidates and added to the current candidates. These will then spawn randomly as the game progress Whenever the player completes a challenge, the function to update the environment is called, which executes the following steps (Code is in the figure below):

1. A challenge is spawned from the current candidates, if there are still candidates left in the future candidates pool.
2. If there are less entries left compared to the amount we want to check it the future candidates pool, the max entry is updated.
3. If the player has completed 5 challenges (*nextLevelTracker == nextLevelTarget*), the current candidates will be updated. (*SwapEnvironment()*)
    a. Remove the oldest entry from the current candidates pool
    b. Get a random index between 0 – *newChallengesMaxEntry* (Step 2)

c. Add this entry to the current candidates and remove it from the future candidates

```
private void UpdateEnvironment() {
    InstantiateEnvironment();
    if (challengeCandidates.Count != 0) {
        if (challengeCandidates.Count < newChallengeMaxEntry)
            newChallengeMaxEntry = challengeCandidates.Count;

        nextLevelTracker += 1;

        if (nextLevelTracker == nextLevelTarget) {
            nextLevelTracker = 0;

            SwapEnvironment();
        }
    }
}
private void SwapEnvironment() {
    activeChalleges.RemoveAt(0);

    int _randomEntry = Random.Range(0, newChallengeMaxEntry);
    activeChalleges.Add(challengeCandidates[_randomEntry]);
    challengeCandidates.RemoveAt(_randomEntry);
}
```

Rubber band effect

The script referenced in this section can be found in
"*RoboRun\CompetenceProject\Assets\Scripts\EnemyBehaviour.cs*"

The enemy wall's movement is determined by the distance to the player. The script containing enemy behaviour checks which enemy it is on. If it is on the Enemy wall the following steps will happen (Code is shown in the figure below):

1. Get the player's sprinting speed
2. Calculate the normalized distance to the player (based on the max allowed distance)
3. Clamping the speed between a minimum and maximum allowed speed
4. Setting the new speed

```
else if (gameObject.tag == "EnemyWall") {
    float playerSprintSpeed = player.GetComponent<PlayerMove>().sprintSpeed;

    float distanceToPlayer = Vector3.Distance(player.transform.position, transform.position);
    float distanceToPlayerNorm = distanceToPlayer / maxAllowedDistanceToPlayer;

    newPosition.z = Mathf.Clamp(
        playerSprintSpeed * distanceToPlayerNorm,
        playerSprintSpeed * minAllowedSpeedScale,
        playerSprintSpeed * MaxAllowedSpeedScale);

    rb.velocity = newPosition;
}
```