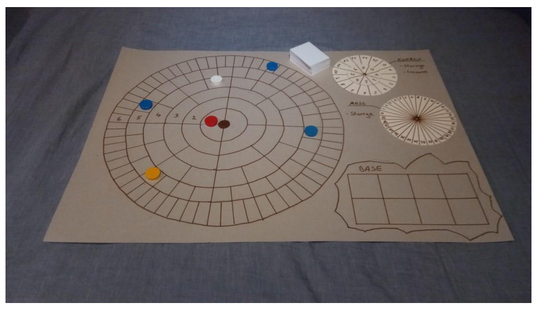
Dark Rip

Erik Franz, Moritz Kohr, Phillip Hohenester

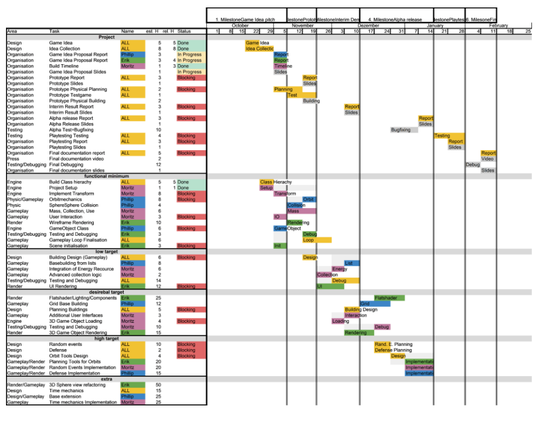
Computer Games Laboratory
WS18/19 , Technische Universität München

Stages of Development

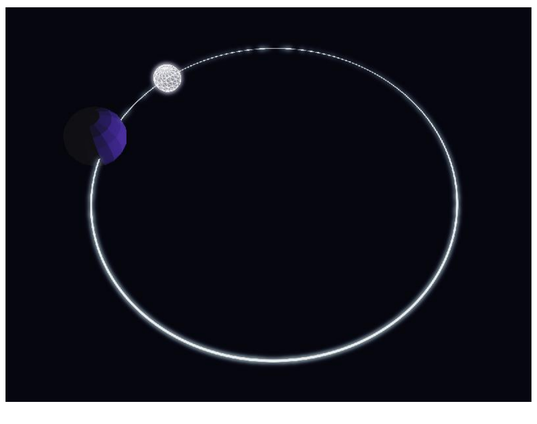
Concept Art



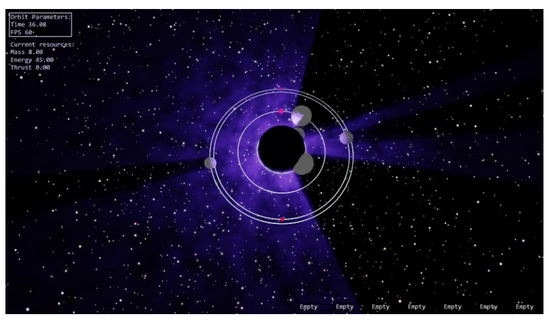
Physical Prototype



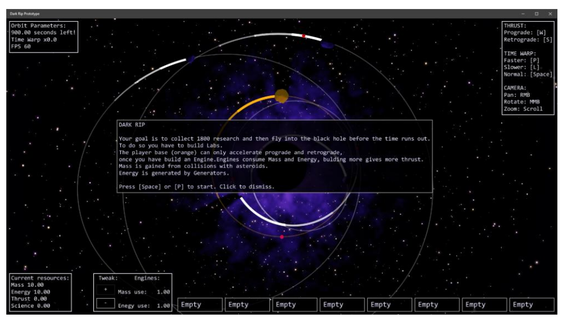
Timeline



Basic Engine



Basic Gameplay



Released Game

Gameplay:

The Universe comes to an end, the Dark Rip is close. Your crew tries to flee from the inevitable. All thats left is your ship and the last place in the universe that still has mass – A Black Hole. Your only chance is to enter the Black Hole and hope that something exists on the other side.

Dark Rip is a short strategy game, you have to:

- Gather Science to safely enter the Black Hole
- Collect Asteroids to gain mass by changing your orbit
- Use Mass to move or construct buildings
- Enter your new universe!

Orbit mechanics:

You may burn Retrograde (against your movement direction) or Prograde (into your movement direction).

Consider that you get higher if you burn prograde, and higher orbits are slower (meaning you actually slow down if you accelerate)

Goal of the Game:

You have to collect 1800 Science within 15 minutes.

After you archived that goal you should enter the Black Hole to avoid dying with the Universe.

The engine is based on DX12

Our Engine

Architecture:

- Based on DirectX 12
- Build on top of UWP
- Uses Entitiy-Component System
  - GameObject as Entity
  - Components as themself
- Split into Subengines:
  - GraphicsEngine (everything that is printed on screen)
  - PhysicsEngine (collisions)
  - InputEngine (User Input, UI)
  - GameMain (time, SceneGraph root, Gameloop)
- currently no sound, multithreading or networking

Graphics:

- Mesh loading from disc
- Simple Material System
- Postprocessing
  - Bloom
  - Antialiasing (MSSA)
  - Render to texture (Orbits)
  - User Interface
- Volumetrics
- Anchors for UI

Input:

- All Virtual Keys mapped
- Touch support
- User Interfaces
  - Buttons with callbacks for clean implementation
  - Wrapper for .Net class

Physics:

- Simple Sphere-Sphere Collision

Orbits:

- analytically solved (very stable)
- 2-Body Problem
- Boxed Orbits possible (still solved for 2-Body problem)
- Component-Based
- Markers for Player

Other:

- Transform Component optimized for performance
- Behaviour Base-Component for easy extension
- Events in all Components (OnStart, OnUpdate, …)