
# Spaceship

## Proseminar Games Engineering

Documentation

**Authors**

| | | |
|---|---|---|
| Florian Anke | 6691498 | fanke@mail.upb.de |
| Alexander Niggemeier | 6681367 | aniggem1@mail.upb.de |
| Dennis Sehring | 7002663 | dsehring@mail.upb.de |
| Lukas Stratmann | 7005026 | lumpiluk@mail.upb.de |

# Contents

# 1 The Game

## 1.1 Installation

### 1.1.1 End Users

Download from Sciebo (includes project files and executables)
https://uni-paderborn.sciebo.de/index.php/s/E9BuBwsgwINEkTU
Password: Games

### 1.1.2 Developers

Clone the project from:
irb-git@git.cs.upb.de:dsehring/team-spaceship.git (https://git.cs.upb.de/dsehring/team-spaceship)
Or download it from sciebo like end users and take the project folder.

Then launch the "Spaceship.uproject" file.

## 1.2 Controls

| | |
|---|---|
| Mouse movement: | Control ship |
| Left mouse button (LMB): | Shoot laser (only near asteroid fields) |
| Right mouse button (RMB): | Use turbo |
| ESC: | Open pause menu |

## 1.3 Tutorial

After starting the game you are given an interactive tutorial which explains the goal and the controls of this game. The real game begins when you successfully completed the tutorial.

## 1.4 Mission

You are taking part in a race in low earth orbit. In your path are layed out an array of space stations with the potential to charge your turbo reserves. Occasionally, you will also encounter small asteroid fields, which are usually artifacts of sloppy asteroid mining. These asteroids you should avoid, since they will slow you down if you hit them.

Use the mouse to control your ship and fly through the correct ring of each space station. When approaching a space station, the four lights in two pairs will flash in short intervals. Fly through the ring between the first light of the first pair, and the first light of the second pair.

You can acquire turbo by flying through space station rings. Depending on how well you are discerning the flashing lights by flying through rings accordingly, you will receive more turbo and hence will be able to finish the race earlier. Your current amount available is displayed at the bottom of the screen. Press and hold RMB to use turbo and boost your ship's speed.

## 1.5 Motivation

The core motivator of our game is it's racing aspect. We assume that a player wants to finish early in order to get placed high on the highscore list. To not lose speed, the player must avoid colliding with space stations and asteroids. Of course, once she has reached maximum speed, the player could hypothetically keep steering to the sides and avoid the space stations altogether. In order to make this scenario less desireable,

the maximum speed that can be reached using turbo is somewhat higher than the regular maximum speed. Furthermore, the player will decelerate back to regular speed once turbo has been deactivated.

## 2 The Experiment

The purpose of this project is to see how well certain perception experiments may be embedded in games and, in turn, how much these games can help to make long experiments less tiresome. The experiment we use for this purpose examines the following hypothesis: Higher-brightness stimuli contrasted against surrounding lower-brightness background stimuli are more salient and will be perceived earlier than said background stimuli.

A space station (cf. Figure 1) consists of four rings and four lights. Two lights in opposing corners form one pair. On each encounter with a space station the player will make two temporal order judgments (TOJs) by opting for a ring to fly through. Presently, only one of these TOJs will include the brighter target stimulus.

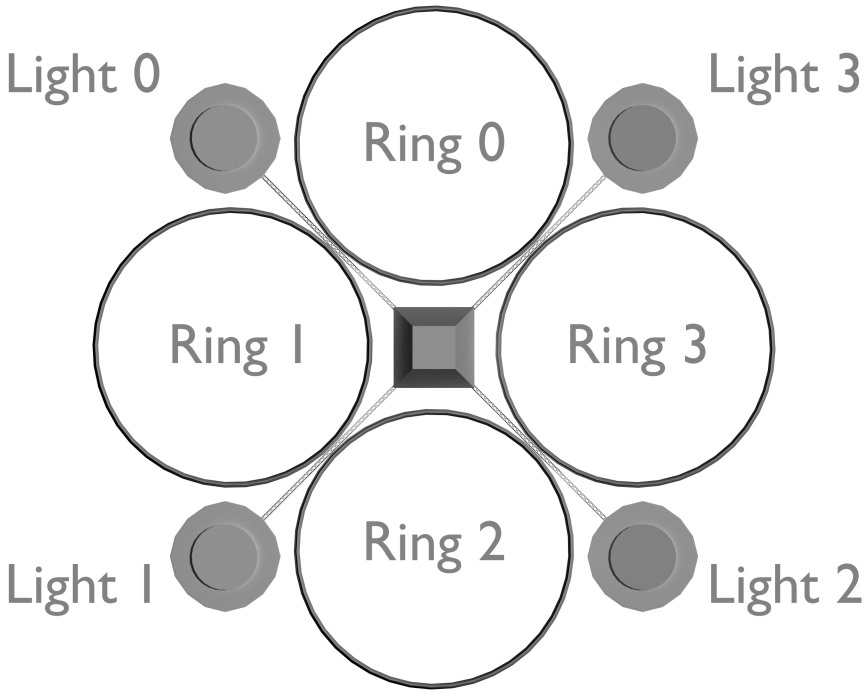


Figure 1: A space station with light and ring indices.

### 2.1 Independent Variables

We used the following stimulus onset asynchronies (SOAs) in our experiment. However, these can be easily changed if so desired (cf. subsection 4.2).

| -100ms | -50ms | 0ms | 50ms | 100ms |
|---|---|---|---|---|

Table 1: SOAs used in the experiment.

Our target luminances can be found in Table 2. The three background stimuli always used the lowest value. By default, the player will encounter 700 space stations in addition to those spawned during the tutorial, resulting in 700 usable trials. In our test runs we managed to finish the game after about 45 minutes.

| | | | |
|---|---|---|---|
| 0.345 | 0.471 | 0.596 | 0.878 |

Table 2: Luminance values used in the experiment.

# 3 Technical Aspects

## 3.1 Required Software for Development

- Unreal Engine 4.10.2 or higher
  - (Our project comes bundled with Rama's Victory Plugin[1])
- Python 3 (for converting log files, Python 2.7 should work as well)
- On Windows: Visual Studio 2015 (at least the community edition)

## 3.2 Tested Hardware

### System 1

**CPU:** Intel®Core™i7-3630QM

**RAM:** 12GB DDR3

**GPU:** Nvidia GeForce GTX 675MX, 4GB GDDR5

**Screen resolution:** $1920 \cdot 1080$ px$^2$ (80 FPS)

### System 2

**CPU:** Intel®Core™i5-4570

**RAM:** 24GB DDR3

**GPU:** Nvidia GeForce GTX 970, 4GB GDDR5

**Screen resolution:** $3840 \cdot 2160$ px$^2$ (68 FPS)

# 4 Project Structure

## 4.1 Important Configuration Variables

- Blueprints/SpaceGameState
  - Asteroids

    **AsteroidFieldInterval:** Number of stations between asteroid fields.
    **AsteroidsToSpawnMin:** Minimum number of asteroids per asteroid field.
    **AsteroidsToSpawnMax:** Maximum number of asteroids per asteroid field.
    **AsteroidScaleMin:** Minimum size of an asteroid.
    **AsteroidScaleMax:** Maximum size of an asteroid.
  - SpaceGame

[1] https://wiki.unrealengine.com/Rama%27s_Vertex_Snap_Editor_Plugin#Plugin_Download

**ShowDebugInfo:** If true, debugging information will be displayed on-screen during gameplay.

**SimultaneousSpaceStations:** How many space stations should be active at the same time.

**StationDistance:** How far away each space station will be positioned from the one before it.

- Experiment

  **ExperimentVarsRepetitions:** How often to repeat each row in Table_ExperimentVars (in random order).

- Tutorial

  **TutorialEnabled:** If set to true, the tutorial will be started with each new game.

  **TutLuminance:** During the tutorial, all stations will use this luminance for all of their four lights.

- SpeedAndTurbo

  **Speed:** The initial speed. Should be equal to SpeedMin.

  **SpeedReductionAsteroidCollision:** How much to reduce the player's speed when the player collides with an asteroid (before it exploded).

  **SpeedReductionStationCollision:** How much to reduce the player's speed when the player collides with a space station.

**ChargingColorCorrect:** The color of charging laser beams when the player flies through a correct space station ring.

**ChargingColorPartCorrect:** The color of charging laser beams when the player flies through a partly correct space station ring.

**ChargingColorWrong:** The color of charging laser beams when the player flies through a wrong space station ring.

**SpeedMin:** The minimum speed.

**SpeedMax:** The maximum speed when not using turbo.

**TurboDepletionRate:** How quickly turbo runs out when being used. In turbo units per second.

**TurboFillCapacity:** Capacity of the turbo reserve.

**TurboBonusCorrect:** With how much additional turbo to reward the player when she passes a correct ring.

**TurboBonusPartCorrect:** With how much additional turbo to reward the player when she passes a partly correct ring.

**TurboBonusWrong:** With how much additional turbo to reward the player when she passes a wrong ring.

**TurboSpeedMax:** The maximum speed when using turbo. Once turbo is deactivated, TurboSpeed will start reducing down to Speed at TurboCooldownRate.

**TurboAcceleration:** How quickly to arrive at TurboSpeedMax after activation.

**TurboCooldownRate:** How quickly TurboSpeed will reduce down to Speed after turbo has been deactivated.

- Blueprints/Actors/SpaceStationActor

  - Movement

    **FlashDistance:** At what distance from the player to begin presenting stimuli (i.e. flashing the lights).

  - Experiment

    **StimulusBgLuminance:** Luminance of the three lights on a station that constitute the background stimuli. By default this is set to the darkest of the luminances in the independent variables.

**FlashFramesOff:** For how many frames a light should be off when flashing until it turns back on again.

**FlashPairDelay:** Time between the two pairs of flashing lights. (first light : SOA $s_a$ : second light : FlashPairDelay : third light : SOA $s_b$ : fourth light).

**LuminanceOff:** Luminance when light is not on.

**ChargingDuration:** How long to keep charging particle systems active after begin overlap.

### 4.2 Overview of Important Blueprints and Assets

- Assets/CSV

  **Table_ExperimentVars:** Defines experiment values like SOA and luminance. Please note: If you wish to add or remove columns, the struct T_ExperimentalVarsInput will first have to be adapted, as well as the corresponding nodes e.g. in SpaceGameState.ExperimentalValuesForNextStation.

- Blueprints

  **SpaceGameInstance:** Saves and transfers values of variables between levels

  **SpaceGameState:** Relevant to the experiment:

  **Experimental Variables** The behavior for reading experimental variables and assigning them to space stations is defined in the following functions: InitExperiment (fills the queue of experimental variables and shuffles it), AddSpacestation (spawns a space station using the top element in ExperimentalVarsQueue for its experimental variables).

  **Logging** Logging behavior is defined in the following Blueprint functions: InitLogFile (look here if you want to change the default log file name and path, for example), LogStation (combines all relevant information in a string to be written to the CSV file as one line). The C++ functions probably need not be touched.

- Blueprints/Actors

  **SpaceshipPawn:** (Actor) or (Pawn) User controlled Spaceship. You can change, for example, the movement of the actor, the drift correction, the invisible cage and the laser action.

  **SpacestationActor:** (Actor) Contains virtually all experimental values that will be logged. A space station will present its stimuli automatically once its location's X component is low enough (as specified in the variable FlashDistance). The functions relevant for flashing are grouped in the category Flashing. PresentStimuli is called first when the space station is close enough. It then adds the light indices in the order determined by SOA and target definitions to the array LightsCurrentlyFlashing and sets the according value in the array LightsOffPendingFrames to the number of frames that this light should stay off. These two arrays will be watched in each Tick event using the function ProcessLightsFrameCounters to eventually turn the lights on again at the right time.

## 5 Results

The game will create CSV log files named e.g. `subject-042.csv` for a player with subject ID 42. The directory where these files are saved will be indicated relative to the game's executable file once a pseudonym and ID have been entered in the game.

Since there are four rather than the common two relevant stimuli, log files might appear confusing to read at first. Essential for evaluation are the columns *StimTargetID*, *StimPair1Bg*, *StimPair1Target*, *StimPair2Bg*, *StimPair2Target*. If and only if the value of any StimTargetID is equal to the value of StimPair2Target, it

means that in the current record the actual target stimulus (the brighter light) was in the second TOJ pair of lights.

The meaning of the values of StimPair1Bg, StimPair1Target, StimPair2Bg, StimPair2Target can be inferred from Figure 1. These four variables indicate which light was used for which stimulus. The variable *Ring* indicates which ring the player flew through, while a value of -1 would indicate a miss. However, while these variables may be useful for debugging purposes, the game already calculates the correctness of the player's decision (*TOJ1Correct*, *TOJ2Correct*), as well as whether the target stimulus has been selected by flying through an adjacent ring (*TargetSelected*).

For information on how to generate raw data summarising files ("TOJ files") from these logs, please refer to the Readme (`README.md` in the project's root directory).

# 6 Discussion

## 6.1 Known Issues

- Progress bar (at the top of the screen in-game) resets to zero sometimes.
- Pressing the space bar rapidly at some points during the tutorial spawns multiple space stations.
- In rare cases, another main menu will appear as a layer behind the current submenu.

## 6.2 Possible Enhancements for the Future

- Figure out a way to create two useful data records from each space station. (Currently only one of the two TOJs is actually used.)
- Additional settings for: Number of space stations, distance between space stations, appearance rate of asteroid fields, default flight speed, delay between blinking and more, customizable save path for result files. Some of these may impact the results of the experiment, so they should be set before each major testing period.
- Add enemy encounters (need to avoid their laser shots) or other racers.
- Make asteroids actually be in the way and maybe harder to shoot (without impacting frame rates too much).
- Levels with different sceneries for more diversity. E.g. different planets, larger asteroid fields, huge structures of large spaceships being built, intergalactic space, under the sea ...
- Add spectator ships lined up along the track, or camera drones.
- Improve realism:
  - More realistic ratio of exhaust flame size vs. acceleration.
  - Thruster placement on spaceship.
  - Orbit.
  - No suddenly appearing or disappearing billboards or space stations or asteroids. (Perhaps work with several levels of detail and spawn at greater distances, gradually increase opacity while still far away, ...)
- Sound design: Don't use music from the Unreal Starter Content in the main menu, create more realistic sound effects for ship-station and ship-asteroid collisions, maybe add a sound effect for shooting lasers.

- Visual feedback when using turbo, e.g. via a larger exhaust flame.
- Improve the tutorial.
- Maybe save the complete game state so that a player may resume a race after having closed the game.

### 6.3 Unreal Engine in Perception Experiments

At first the Unreal Engine can be overwhelming with its multitude of tools and features. Nevertheless, especially thanks to the video tutorials, the core functionalities were surprisingly easy to grasp.

The Unreal Engine certainly seems like a useful tool for creating perception experiments. Our trial runs (which can be found in the project folder under doc/sample_results/) show that a game created with this engine can produce at least superficially reasonable-looking data. Whether or not the engine holds up to the strict timing requirements necessary for experiments like these will have to be seen after a sufficient amout of participants' data has been evaluated.

If it turns out that, indeed, perception experiments work with the Unreal Engine, there is still the question of effort and utility. In case subjects report that our games are more fun than the usual experiments by a big margin, one might consider creating a game in which can be integrated many kinds of perception experiments. The implementation of a new experiment in this hypothetical game would have to be possible in an amount of time comparable to developing the same experiment in e.g. OpenSesame. Another problem might be hardware requirements. Not only do these games need a modern graphics card and CPU in order to achieve frame rates above 100fps, each game also may easily require multiple gigabytes of storage. This latter problem could at least be partly eliminated if a single game or few games were used that can be extended frequently and easily.