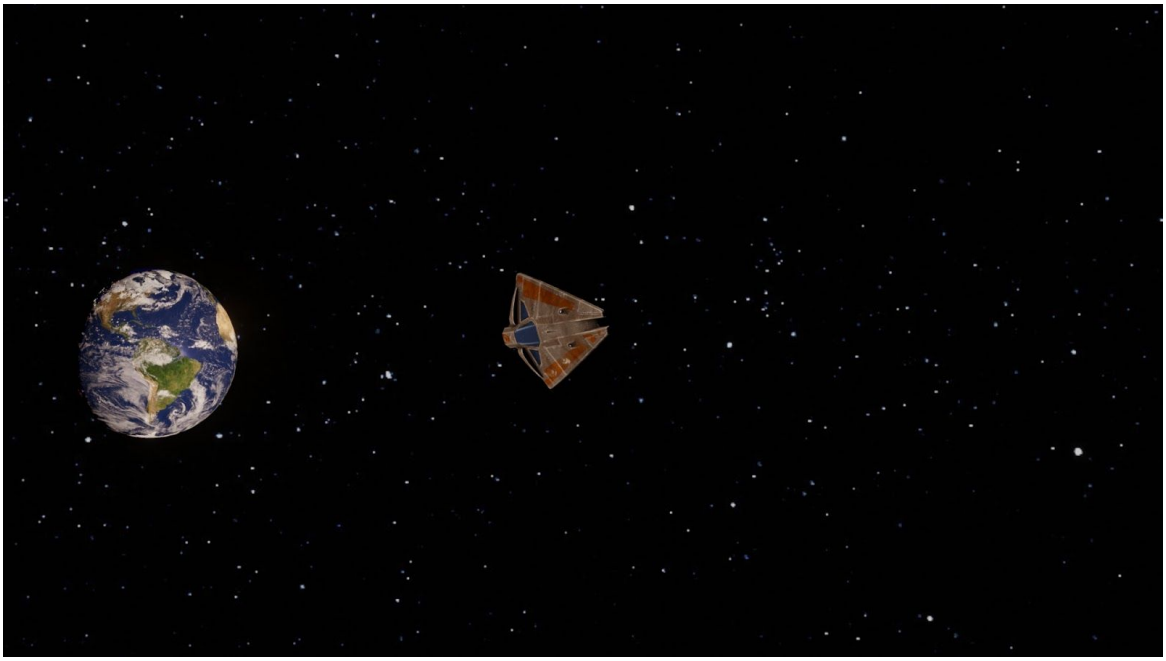
LOST IN SPACE

(Working Title)

Software Project Specification Document



Ondrej Čakloš
Denis Iudin
Vojtěch Řehák
Sebastian Schimper
Dr. Tomáš Holan (Supervisor)

13.09.2019

FACULTY OF MATHEMATICS AND PHYSICS
CHARLES UNIVERSITY PRAGUE

TABLE OF CONTENTS

1. INTRODUCTION

1.1 SURVIVAL GAMES

During the last few years, the computer game genre of "Survival Games" has gained a certain popularity amongst players worldwide. The focus of these games is on collecting items and resources, their reasonable management, exploration and the crafting of items using other items. All this ensures, like the name of these games suggests, surviving.

To name a few examples: In the humorous game "Don't Starve", the player has to make sure to keep the protagonist, a cartoon scientist stranded in a strange world, alive as long as possible by finding food and other items. The more serious game "This War of Mine" transforms the player into several civilians during wartime and lets the player search for food and medicine, craft useful tools and make difficult decisions.

What is fascinating about "Survival Games" is the strong player experience and their "replayability". All the decisions the player makes have an impact on the game, some decisions can be fatal and may set the course to ultimate failure. And when these games are played again, the gaming experience can be completely different.

1.2 MOTIVATION

During the "Game Development" course during the winter semester 2018/19, the first draft of the game "Lost In Space" was being developed. It is a 2D survival game, where the player controls a small spaceship, which got lost somewhere in space, in order to get back to Earth. On the way through different galaxies the player can find planets and scan them for resources, he/she can craft new resources from other resources, and he/she has to dodge asteroids and avoid coming too close to the sun.

You can find a downloadable version of the game on:
https://drive.google.com/file/d/1QOzYAeVLU4lvy4oCRlUcssi1Ad1-yDGP/view

Although a lot of time and effort were spent on this project, the development time was limited and some features could not be realized. Furthermore, the game has the “look-and-feel” of an amateur game.

The motivation for this software project is the continuation of the development of this game, that comes as close to a state-of-the-art indie game as possible.

1.3 THE GOALS

For this software project the team members will take an already existing version of the game “Lost in Space” and expand it with numerous features which will (hopefully) result in a greater player experience. The ideas, themes, mechanics etc. of the game are nothing new, because revolutionizing gaming with groundbreaking ideas will not be the focus. The goal of this software project is to obtain as much knowledge and experience in Game Development as possible, to be allowed to experiment and to develop a game that does not give the impression to the player that it is “just” a school project, rather than a professionally developed indie game. We may have to spend money during the development for buying 3D assets from the internet or licences for the soundtrack music. However, the money spent for this should not exceed a budget of 500 CZK.

2. PROJECT DESCRIPTION

In this section we will provide an overview over core ideas and concepts the team members developed for the game “Lost in Space”, more precisely the setting and “mood” of the game, the desired player experience the team wants to achieve as well as a list of main tasks, which are described in more detail in the “Development”-chapter.

2.1 SETTING

Near future. The living conditions on planet Earth heavily decreased due to climatic catastrophes and a shortage of resources. The idea of resettlement from Earth to another planet brought the smartest scientists together to work on a hyperdrive system which allows for interstellar travels. After a number of successful tests, it was time for a group of astronauts to use this hyperdrive system together with their spaceship in order to explore potential locations that can serve as a new home to inhabitants of Earth. However, during the travel through the hyperdrive, something went horribly wrong and the spaceship finds itself stranded somewhere in the dark vastness of the universe. The top priority for the crew on the spaceship is to — somehow — find a way back to Earth and also to survive on the journey back home.

2.2 PLAYER EXPERIENCE

The player is a skilled spaceship pilot who is stranded somewhere in space, has to navigate the ship and eventually find a way back to planet Earth. He/she has to take care of the spaceship crew by scouting food or medicine items in order to keep them healthy., has to craft items to repair the broken ship and has to face arising dangers in order to survive.

2.3 MAIN TASKS

The following figure displays a so-called “Onion Design”. It consists of a core, which represent the core feature without the game would not work at all.

The other layers represent features which are built on top of the core and also support it. The smaller the distance to the core, the more important is the feature.

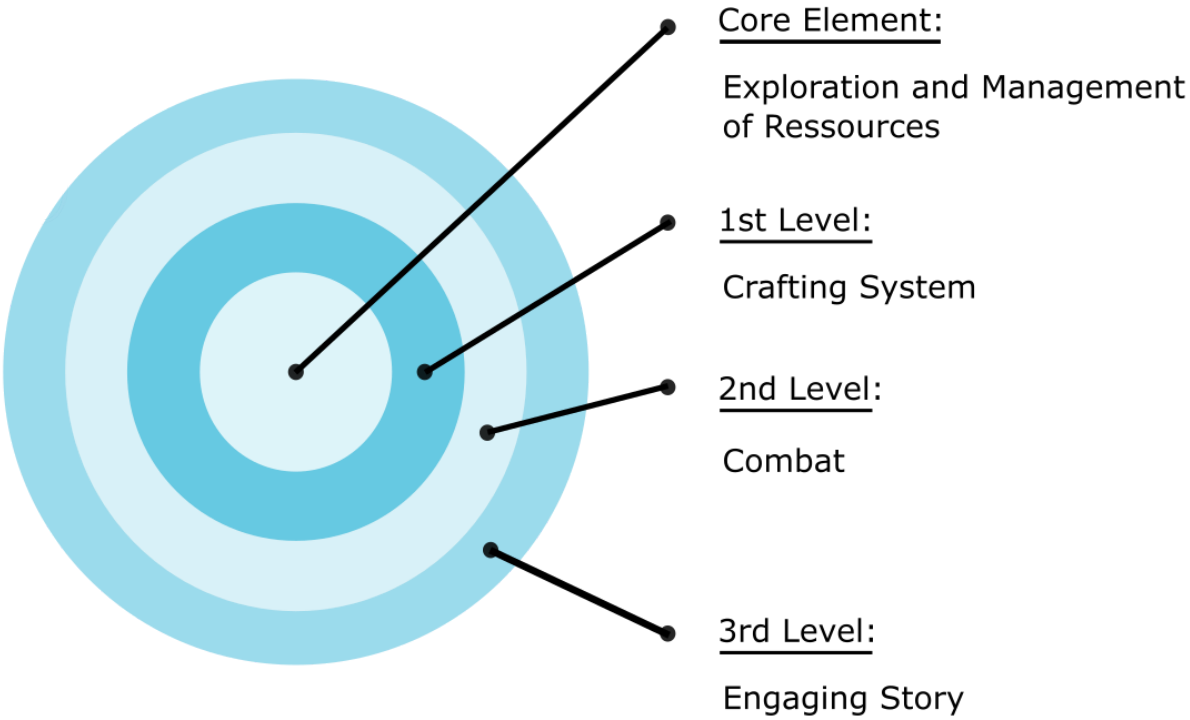


Figure 1: Onion Design

2.3.1 Core

The main aspect of our game is exploration and navigation through an unknown environment. In order to do so, resources like food or gasoline must be harvested. Therefore, the meaningful management of resources is a further core aspect of the game. The core features are already more or less present in the game, thanks to the development during the “Computer Game Development” course. Nevertheless, an improvement of these features is desired for the software project (e.g. more items, a greater environment).

2.3.2 1st Layer

This feature directly expands the core feature. If certain items are found they can be

combined to craft other important items. During the project, a crafting system will be implemented for crafting things like a canon for the ship for defense purposes or a device which allows for communication to Earth or other intelligent life.

2.3.3 2nd Layer

Combat. The player should be able to fight enemy units and overcome other dangers. Furthermore, the player eventually will be in contact with other intelligent lifeforms. These might be friendly or hostile.

2.3.4 3rd Layer

If there will be enough time, the team members of the software project will try to give the game a cinematic touch. This requires an opening cut scene, screenplay writing and voice acting. The downside is that this is a lot of work if done correctly, but in case these goals cannot be achieved, they are not as necessary as the other described features.

Figure 2: Frame from opening cutscene. (Smoke effect has to be worked on)

3. DEVELOPMENT

This section addresses on the more technical details of the development of the game. First, information about the distribution of the roles amongst the team members and the corresponding tasks is shared. This subsection will also consist of a challenge estimation of the individual tasks.

In the next section discusses how these individual tasks will sum up during the development process.

Finally, the number of software that comes to usage during the development process will be listed, followed by copyright information about certain media we are using.

3.1 THE ROLES

3.1.1 Backend, Game Mechanics - Ondrej Čakloš

The responsibility of this role is the creation of core systems, like inventory and items, which are necessary for the game. Therefore, the first criteria of these systems will be functionality. The systems should also be easy to use and expand.

3.1.2 Realistic Physics, Combat System - Vojtěch Řehák

This role is responsible for creating realistic physics behaviour. This means that every object in the game will have a certain amount of mass. The forces will move with the mass. With these forces will be possible to simulate gravity field and orbits around the planets. The mass and velocity will also affect collision in realistic way.

Second responsibility is creating combat system. That includes enemy's combat and non-combat behaviour. Main non-combat behaviour is focus on some random moving in

space but it has to dodge dangers like Sun, asteroids, etc. The combat behaviour is focused on chasing the player, dodging player's projectiles, using weapons, shields, ...

Last responsibility is generation of asteroids and enemies in some reasonable way. That means reasonable position, frequency of spawning, maximum amount, etc.

3.1.3 User Experience, Level Design, Sound, Asset Modelling - Sebastian Schimper

The responsibilities of this role include — amongst other things — the development of a user experience that is as possible as can be. This means ensuring an easy and intuitive way to navigate through the game by creating a compact and visually appealing User Interface, designing and implementing meaningful controls, design of the levels, Sound Engineering and the modelling of assets that will be used in the game.

3.1.4 Random Generation, Script Writing, Quests - Denis Iudin

This role is responsible for creating a system for random generation of playing space (solar systems, including luminaries, planets, asteroids and other celestial bodies, including man-made objects), as well as creating a quest system (tasks that prompt the user to a particular action). In addition, this role is responsible for writing the game script and thinking over the game universe.

3.2 TASKS AND CHALLENGE ESTIMATION

In this section the main tasks and challenges will be listed, as well as an estimation of the time needed to implement them.

3.2.1 Realistic Physics

At the current version of the game, developed during the “Game Development” course, there are no signs of physics at all. Every movement is based on a simple matrix transformations (rotation, translation, …) or vector addition.

The main goal of this task is to make behaviour of every object in the game as realistic as possible. That means assign mass to every object. With mass comes gravity. Planets have huge amount of mass so gravity of the planets will affect the ships and asteroids passing by. We cannot use built-in gravity force because its direction follows one of the axis of world coordinate system. So every planet will compute its own gravity which will affect objects in its radius. This way we are able to simulate orbits.

Every ship in the game has engines. These engines will create directional force which will move or rotate the ship. Ships also will be equipped with weapons. Ship objects will implement methods

- Forward()
- Backward()
- TurnLeft()
- TurnRight()

Every method apply to object force in correct direction. With these simple methods we are able to create more complex paths that the object e.g. enemy ship can follow.

Asteroids will be generated with some initial force, so they can fly through space.

Realistic physics also includes collisions. So mass, size and speed of moving object will affect the way how the objects will collide. In fact there are just a few options how the objects can collide with each other in our game:

1. Object pushes the other
2. Objects bounce off each other
3. Object damage/destroy the other
4. Object damage/destroy itself

For all these features will be used Unity 3D physics engine which has quite neat interface. Unity 3D offers class Rigidbody. Assigning Rigidbody component to objects makes them affected by gravity, forces, etc… Second important component is Collider. It is bounding box for object. It can have general shape but for quick calculations it is usually shaped like a sphere, cube, etc… When these bounding boxes intersect each other, it invokes collision event.

3.2.2 Inventory System

During the game, the player collects a number of items and these should be displayed in the inventory view. This described task is the design of the inventory. Figure 5 shows an early prototype of the inventory view. On the left half, a panel which contains a grid will appear with all the items that have been collected so far. The right half shows a closeup of the spaceship, the circles indicate parts of the ship that can be updated (boost, guns etc.)

Unlike basic inventory system currently in game, this system will consist of grid inventory capable of storing items of different sizes. Inventory will not be a permanent part of GUI but can be opened and closed. Also, the storage of ship will consist of few smaller inventories instead of one combined.

Items in the inventory can be moved by drag and drop system. During dragging, grid cells will be highlighted, showing if item can or cannot be dropped there. Pointing on an Item will display a tooltip with description of that item.

The inventory system will be made from two classes. One which will take care of basic manipulation with inventory like adding and removing items. The other will care about displaying inventory and interactions with it.

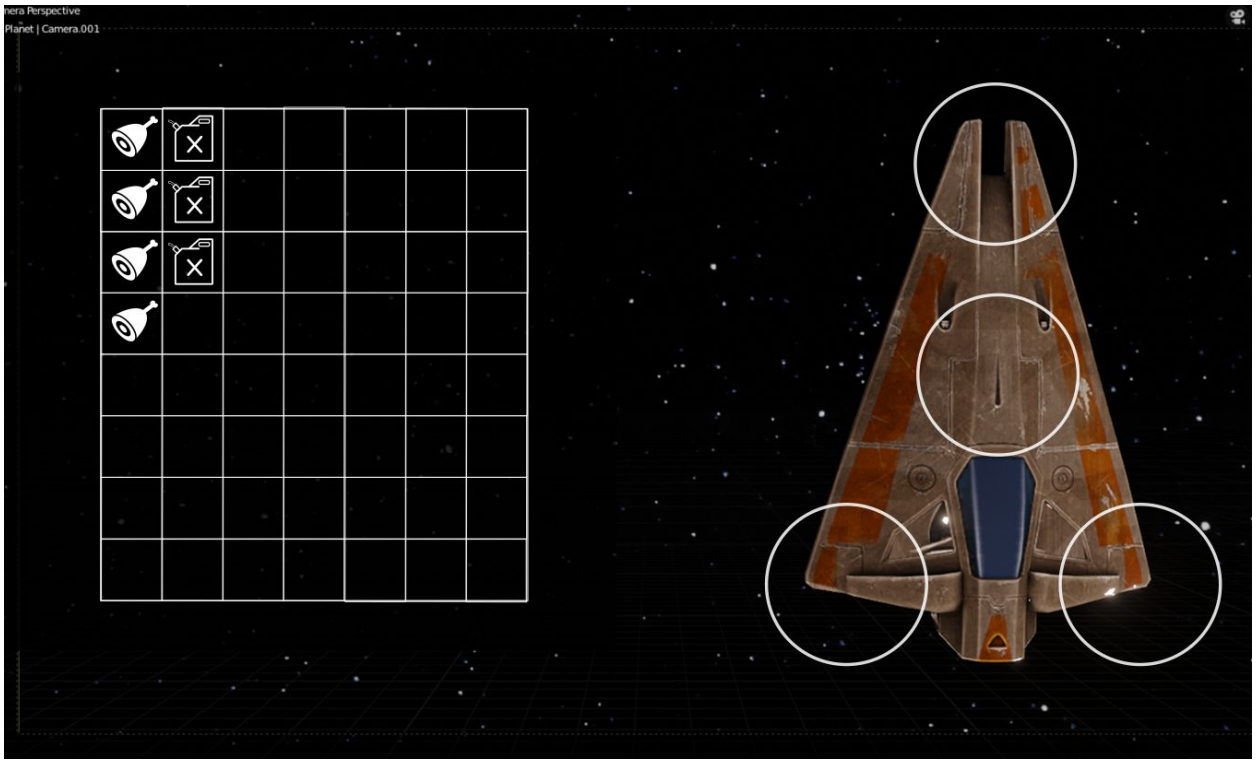


Figure 3: Prototype of the inventory View

The Inventory class will store information about items inside and provide methods to work with it.

- GetItems() - returns a list of all items inside inventory.
- HasItem() - returns true if item is in inventory, false otherwise.
- AddItem() - store new item inside inventory.
- RemoveItem() - removes item from inventory.
- OnInventoryChanged() - Event dispatcher for changes in inventory.

InventoryPresenter class will have two main methods: ShowInventory() and HideInventory(), which will show or hide inventory GUI respectively.

3.2.3 Item System

The item system is one of the core features. Items in this game will take place mostly as resources that are used to craft items essential for players survival. They can have

different sizes and types.

Items in code will be represented by class Item and corresponding subclasses for different types of items such as UsableItem or ShipModuleItem. For better development, creating and updating, we will use benefits of scriptable objects in unity.

Usable items will have boolean variable Consumable. If this variable is set true then items will disappear after the item is used.

Effects of item usage will be defined as class and also will be scriptable object. This way it's easy to add/remove effects or change any effect of an item or multiple items that use the same effect.

A further responsibility will be the design of icons representing these different resources. In Figure 3 two icons are representing food- and gasoline resources.

The time estimated for this task, including the creation of sketches, prototypes and the discussion of them amongst the team members will be **2 days**. The software used for these tasks will be Photoshop and the UI Designer that is included in Unity.

3.2.4 Background Environment in the Game

One major challenge of the project will be the creation of a dynamic background. In the version developed during the "Game Development" course this was accomplished by tessellating 2D-Tiles. These were fine for just one level, however, since an expansion of the game to multiple levels is desired, the background consisting of these tiles will become monotone and boring. A variety of different background images is desired.

The idea is to use images provided by NASA (see Figure 4 for one example) increase the resolution and "polish" the image with software like Adobe Photoshop and cut the image in same-size tiles. The background will be automatically created by using these tiles as Wang tiles as the ship navigates through space. One negative aspect of this is that due to the usage of 2D tiles, the background will look rather flat and will be missing "depth".

The time estimation for this task will be about **7 days**, including experimenting and research.

Figure 4: Image of the NASA gallery

3.2.5 Combat System

The combat system is a completely new element in our game. So far there are no enemies, weapons, fighting, etc. Including these things to the game make the game more fun to play and more challenging. Also, it’s a survival game and combat system adds another layer of survivability.

At the beginning of the game, there will be few numbers of enemy ships generated, equipped with some weapons. At first the player's ship will be unarmed so the only way to survive is to escape the enemy. During initialization of the game, every enemy ship

obtain few key points in the space which one by one will define the moving trajectory of enemy ships.

Enemy ships will have two states represented as bool variable in enemy object. Combat-state and Non-Combat-state. While in non-combat state, the ship will fly through the space following predefined trajectory. This trajectory will be stored as an array of points in enemy object. The AI will guarantee that the ship will not collide with the dangers like Sun or asteroids.

When the player reaches the threshold radius around enemy, stored as float number also in enemy object, enemy ship switches to the Combat-state and start chasing player or start firing projectiles. Or both. Depends on position. Chasing will last until the player is still in radius or until elapsing pseudo-random amount of time. Enemies will also be able to make evasive maneuver against incoming projectiles or objects like asteroids.

As the player will obtain more and more resources while playing then he can craft and improve his ship among other things with weapons and shield. These gadgets help player to be more powerful in combat. For example, there will be multiple types of weapons like cannons, lasers, machine-guns, etc.

There will be base class CombatItem. Child classes will be OffensiveItem and DeffensiveItem and then particular items.

Every object will have a certain amount of health. Damage caused will be based on the physical system. That means it will be computed from speed and mass of the object/projectile.

3.2.6 Crafting System

Crafting is an essential part of survival game. Crafting will be separated into several tiers. For crafting higher tier items you need to create better crafting module. Progressing through the crafting tree should take some time and effort.

Recipes for crafting will be represented by class with scriptable object. This will allow us to create and modify recipes quickly and easily.

Crafting Presenter will be a class which will take care of GUI. Methods ShowCrafting and HideCrafting will open or hide crafting panel respectively. Panel should show only

recipes that can be crafted with current modules. Recipes player don't have resources for will be grayed out (but visible).

3.2.7 Ship Modularity System

Players ship will consist of multiple parts where every part will have special function. There will not be any concrete way how to build the ship. In other words, modules or ship parts can be connected together as player likes. This will create some decisions to make through the play, where the player will need to decide where to put which part to get the best result.

As the ship can be built part by part, it can also be destroyed in the same manner. This means that every part of the ship will have health points and if ever these points reach zero the part will be destroyed.

Modules can be found in three types. Firstly as a part of the ship, already build and providing functionality to player. Secondly as an Item, inside inventory and ready to be added to the ship. And lastly as a crafting recipe, which will create an Item if appropriate resources are available. In this section we will discuss only the already built part of the ship variation.

The modul will be represented as class with scriptable object. Similarly to UsableItems effects, modules functionalities will be another class with scriptable object for best possible development.

3.2.8 Procedural Solar Systems generation

The process of procedural generation of a new solar system will start when the user moves from one scene (system) to another. It can be divided into two large components: planet generation and the generation of the solar system as a whole.

Planet Generation:

To generate a planet, a class describing the planet will be needed. That is, a class

containing information about all possible types of planets, possible types of resources on them, the possible number of these resources, as well as many other parameters, including methods for modifying the planet object on which this class will be suspended in the form of a script in the editor of Unity.

Then you need a class that is directly responsible for the generation of the planet. Going a little further, we will also need a class to generate the system, and since all this is interconnected, I decided to combine these two classes into one.

When creating a new object of the Planet type, the CreatePlanet() method is called, which randomly places the given planet in space. Then it determines the distance from the placed planet to the sun and calls the SetPlanetType() method, which, depending on the distance to the sun, determines the type of planet (one that is too close will be molten and too far will be ice). Then the PlacePlanet() method is called, which is responsible for determining the planet's resources depending on its type (specific resources are tied to specific types of planets and will not be found on others, for example, water cannot be found on a molten planet, but what available resources will appear on the planet and what there will be their number — determined randomly), then this method physically places the created planet in space.

Solar System Generation:

The generation of the solar system is based on a number of parameters, such as the number of suns in the system, the number of planets (in general), the number of asteroids in the system, and other similar parameters for all possible celestial bodies. Their number is determined randomly, although the game designer has the opportunity to manually set the boundaries of the ranges for generation, including setting which ones he wants them to be in this system. For this, a system known as "grain" will be used. The grain can be randomly generated or set manually. For example, in the grain it will be said that in this system there must be 2 ice planets — in this case a two will be added to the number of planets in the system and two planets will be generated not in a completely random coordinate range, but in a range corresponding to ice planets.

Based on a randomly generated number denoting the number of celestial bodies of one type or another, an appropriate number of objects with random characteristics will be created.

3.2.9 Game Mechanics

In this section we'll discuss what game mechanics and how they are influencing gameplay. Each of these mechanics is a combination or result of multiple systems described above. Any time estimation in this section will only be concerned about tweaking and polishing of that mechanic.

- Exploration - Main mechanic of this game is exploration. We want to player to wander through the space not knowing what he can find. Supporting features for this mechanic are: Limited line of sight, randomly generated map and objects player can encounter. The overall estimation will be around **30 days** separated throughout the development.
- Resources - Resource management, mining and crafting are basic tools for survival games. Planets can be mined, asteroids can be gathered to get necessary resources for survival. It should take **7 days** to create.
- Dangers - Are objects that can be found in space, but can be harmful to player. This will create some tension to the game where player will need to decide if reward outweighs risk. Possible dangers are being too close to the sun, enemy ships, asteroid belt, asteroids and other. The time estimation **14 days** separated throughout the development.
- Progress - Starting with simple ship which can barely fly and progressing to fast traveling cruiser that can safely take you home. Progress is about upgrading ship, crafting new materials and eventually repairing hyperdrive. Tricky part will be to get the right speed of progress so player is not bored by game being too easy for his mighty vessel or work hard to get only miniscule reward out of it. Estimating this mechanic for **10 days**.

3.2.10 User Interface

Upon starting the game, the main menu will be the first thing the player will encounter. He/She is presented following options: Continuing the game from the last saved point. This is only possible if the game has not been started for the first time and if there is some saving data. The player can furthermore start a new game. If there is already

saving data available, a pop-up dialog will appear asking informing the player that all progress will be lost if a new game is started. If the player wants to start the game from a point where he previously saved the game, he can load his/her current game stand. Finally, the player can find some information about the team behind the game and, of course, he/she can quit the game.

When the player plays the game, he/she can pause the game by pressing the Escape key, which will result in a pop-up pause menu. From here, the player can save his/her progress, can load a different game, can exit the game to the main menu or quit the game entirely.

The development of the main menu is more or less done, what is missing is the design and implementation of the pause menu. The UI design task intersects with another tasks, namely the implementation of a Load/Save-System which is described in the next subsection. The creation of Load/Save-slots in the UI will be excluded from this time estimation. In addition to this, several UI-elements that are displayed during the game will be needed, such as a health bar and a navigation panel.

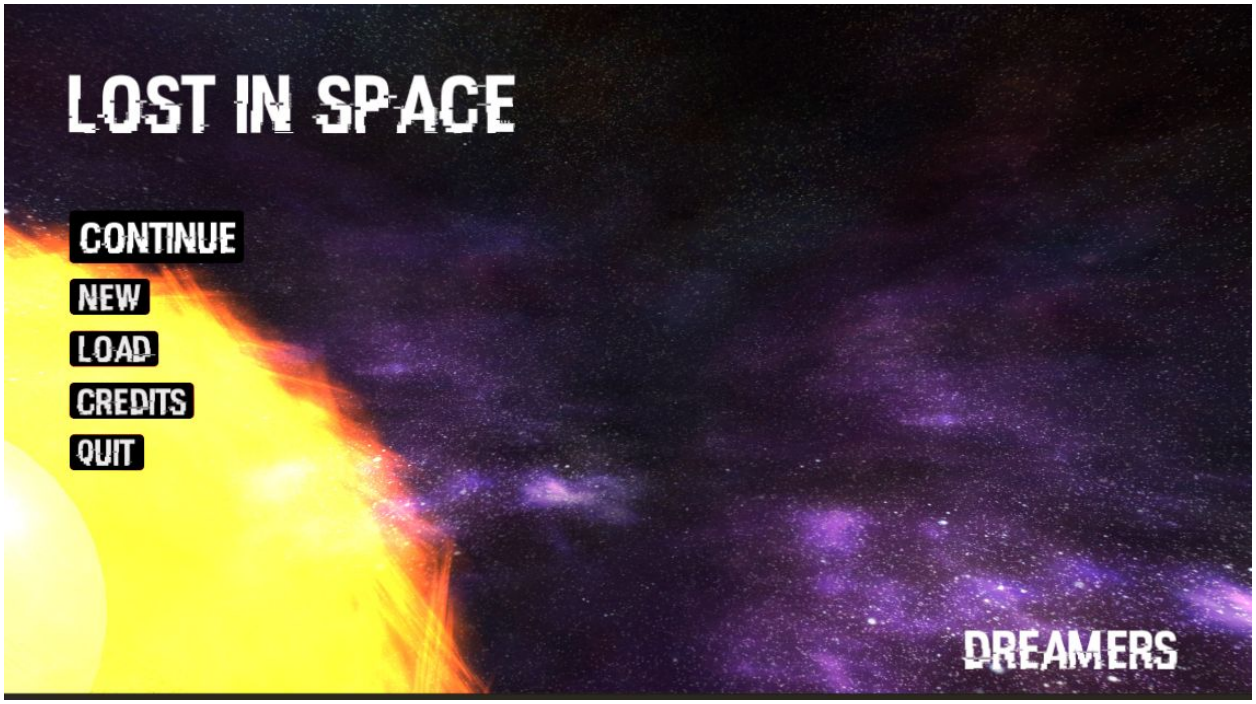


Figure 5: Main menu

3.2.11 Load/Save-System

Whenever the player is playing the game, he/she should be allowed to save his/her progress and continue from there at a later point in time. The saving of a game progress will result in the creation of a UI-element which we will call “Slot” for convenience. These slots will be displayed in the “Load Game” submenu of the main menu and ordered by their timestamp. In the main menu, the “Continue”-button will automatically load the most recent slot. In the pause menu, when saving the game progresses, the player can choose to create a new slot or override an existing slot. Furthermore, a functionality will be implemented that automatically saves the game progress every 15 minutes.

3.2.12 Story

Story is not the main part of this game, the gameplay (mechanics) is brought to the forefront, but this part, narrative, is still important, while being the least "technical" of all. It gives the goal and motivation to the player, explains what is happening, connects the mechanics together, gives depth.

Story in this case means the following:

- The main quest is a combination of several quests connected by a single logic (scenario), designed to lead the player to a logical conclusion to the game, satisfying the conditions of "victory".

- Side quests and events - each of which has its own small scenario, different from the others, and usually does not affect the others (although there are quest chains, and for the beginning of some, the completion of other quests is necessary).

- Development and description of the game universe — necessary for the authenticity and awakening of a player's interest in exploring the game. It is not demonstrated

directly, but it is a kind of design document right inside the game: who inhabits this world, what kind of relationship they have, what happened in the past. The player will encounter this simply by interacting with certain non-game characters (they didn't come from nowhere, each has a story that determines his motivation), fighting opponents (they have reasons to be aggressive), finding artifacts in space, etc.

3.2.13 Side missions and events

Side missions are tasks received by the player from the NPC (non-player character) in order to induce him to perform certain actions (since systems are randomly generated, non-player characters will also appear in random places). Also, the activation of a side quest can be caused by the fulfillment of certain conditions: being in a certain place, completing another quest, etc. The incentive is both the ultimate reward and the process itself. Side quests consist of a narrative: texts in which the player is told what needs to be done and why it is needed, on behalf of a non-player character or player character. And also from the part of the actions, which depends on the type of quest: in text quests, the player needs to choose the options from the ones presented in the list, while other quests require a "physical" action: to be in a certain place, to perform a certain action with a specific object, etc.

Each quest will be implemented as a separate script (class), as this allows them to be configured and worked out more flexibly. Quests will contain activation conditions, a list of actions required to complete (step by step) and verification of their implementation, a list of rewards (and the initiation of their transfer to the player), as well as a link to lines in the csv-format file containing the necessary text to which he will refer and use. This approach is convenient in itself and, if necessary, greatly simplifies the creation of localization in another language.

Events differ from quests in that they do not imply any action on the part of the player — when the activating condition is satisfied, the event occurs and immediately causes consequences: it activates a new quest, improves or worsens the player's position.

3.2.14 Sound and Music

The sound effects used in this game will be sound samples from the website “freesound.org”. The website provides a broad database of sound effects which can be downloaded and lay under the creative-commons license.

For the background music, working with the software “Magix Music Maker” will be considered. Music Maker provides a set of recorded sound samples, which can be composed to a music track. These sound samples are free to use for non-commercial purposes.

The goal here is to compose 4 to 5 different background music tracks. The time estimated for this task — including training with the unfamiliar software Music Maker — will be **3 to 4 days**.

3.2.15 Modelling of further Assets

As the scope of the game will grow during the development, there most certainly will be the need for further assets. These can be different planets, alien spaceships, asteroids etc. One option is to download existing and free-to-use-models from websites like “TurboSquid”, but if no suitable model can be found, models need to be created manually.

Crucial assets needed for the game will be planets, which are basically spheres with different shaders applied to it. More challenging would be the modeling of spaceships, moreover, the player’s spaceship. Different models of the ship representing the states of damage need to be created as well as ships with crafted features like guns etc.

As this task will concretize during the developing path, a binding statement about the estimated time cannot be given yet. The only thing that is certain is the fact that this responsibility will take at least **15 days**.

3.2.16 Opening Cut Scene

If the time allows for this, a cut scene to introduce the player to the game and to add a little cinematic touch, can be directed. Since the animation progress with take some time for the inexperienced team members, this task is not obligatory because the opening cut scene is not crucial for the gaming experience.

3.3 DEVELOPMENT PHASES

In this section a visualization of the specific tasks and the estimated time needed to complete them for every team member.

The project officially started on the 8th of August 2019 and the final deadline for it will be 9 months later starting from this point.

Basically the development of our game can be broken down into three different main phases:

- The Preparation Phase (8th Aug, 2019 - 30th Sep, 2019)
    - Development of ideas and concepts
    - Planning and Scheduling
    - “Paperwork” and other administrative tasks
- The Development Phase (1st Oct, 2019 - 20th Mar, 2020)
    - Implementation of the Concepts and Ideas
    - Testing
    - Releasing Alpha- and Beta-Version
    - Can be broken down into subphases, e.g. Development before Alpha release, between Alpha and Beta release and after Beta release
- The End Phase (21st Mar, 2020 - 21st Apr 2020)
    - Writing of the Documentation
    - Additional administrative tasks

Figure 6 shows the iteration process for the implementation of each task. The team members will meet in person at least once every week to discuss the results of every team member and to give feedback.

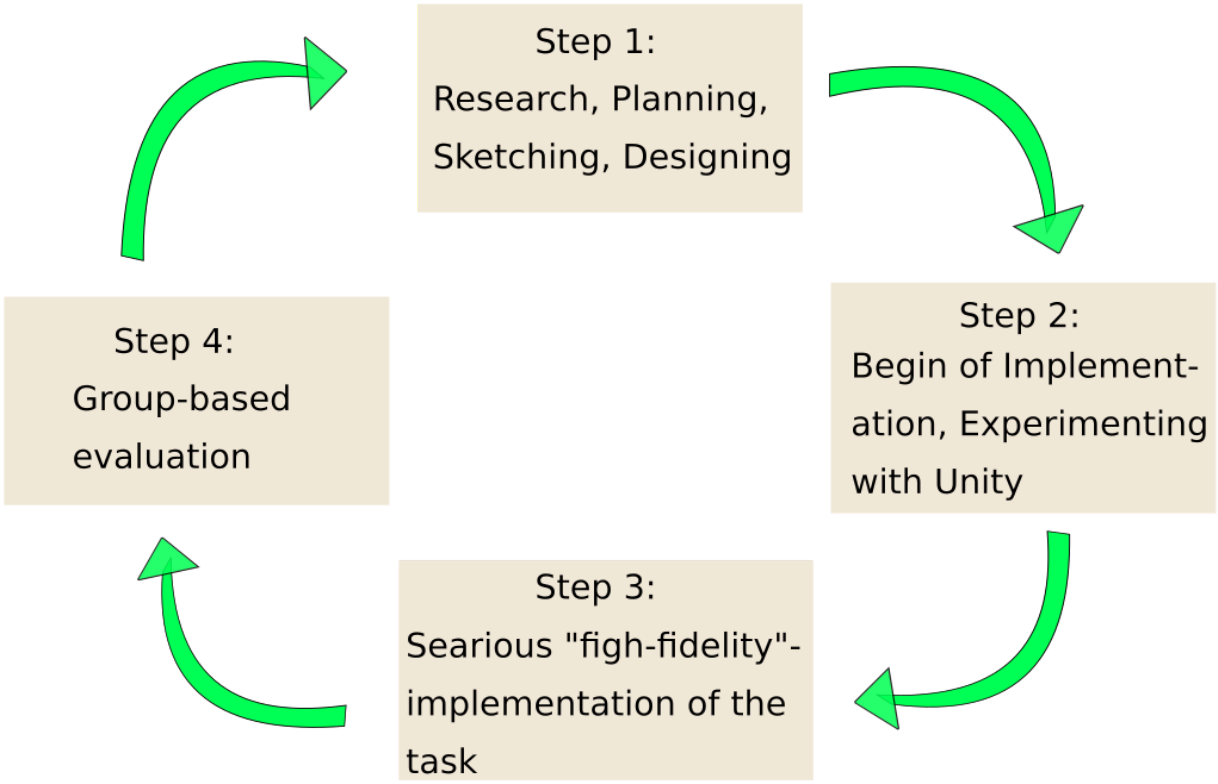


Figure 6: Iteration process of each task

Note that one can not only from step to step in a circular way. In fact one can jump from any step to any other step in the diagram. We believe this will help us to get the most out of our work.

The following figures will visualize the “Development Phase”. Every task for every team member is shown in a “Roadmap” diagram.

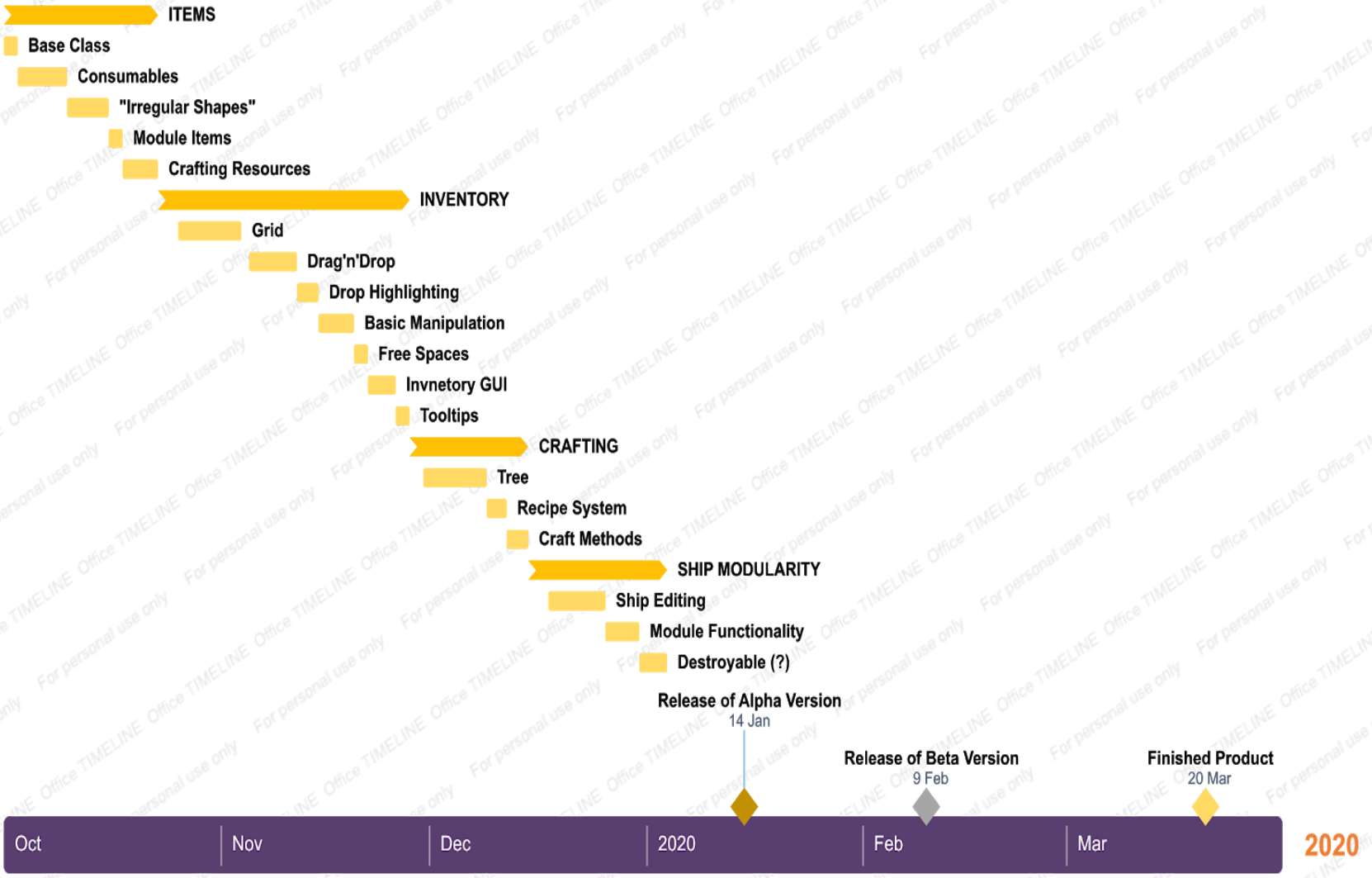
The diamonds on the timeline in bronze, silver and gold will represent the milestones, namely the release of the Alpha-, Beta- and Final Version of our game.

Every color is representing a different team member. Ondrej’s responsibilities are marked yellow, Vojtěch’s are marked purple, Sebastian’s are marked blue and finally Denis’ tasks are marked green.

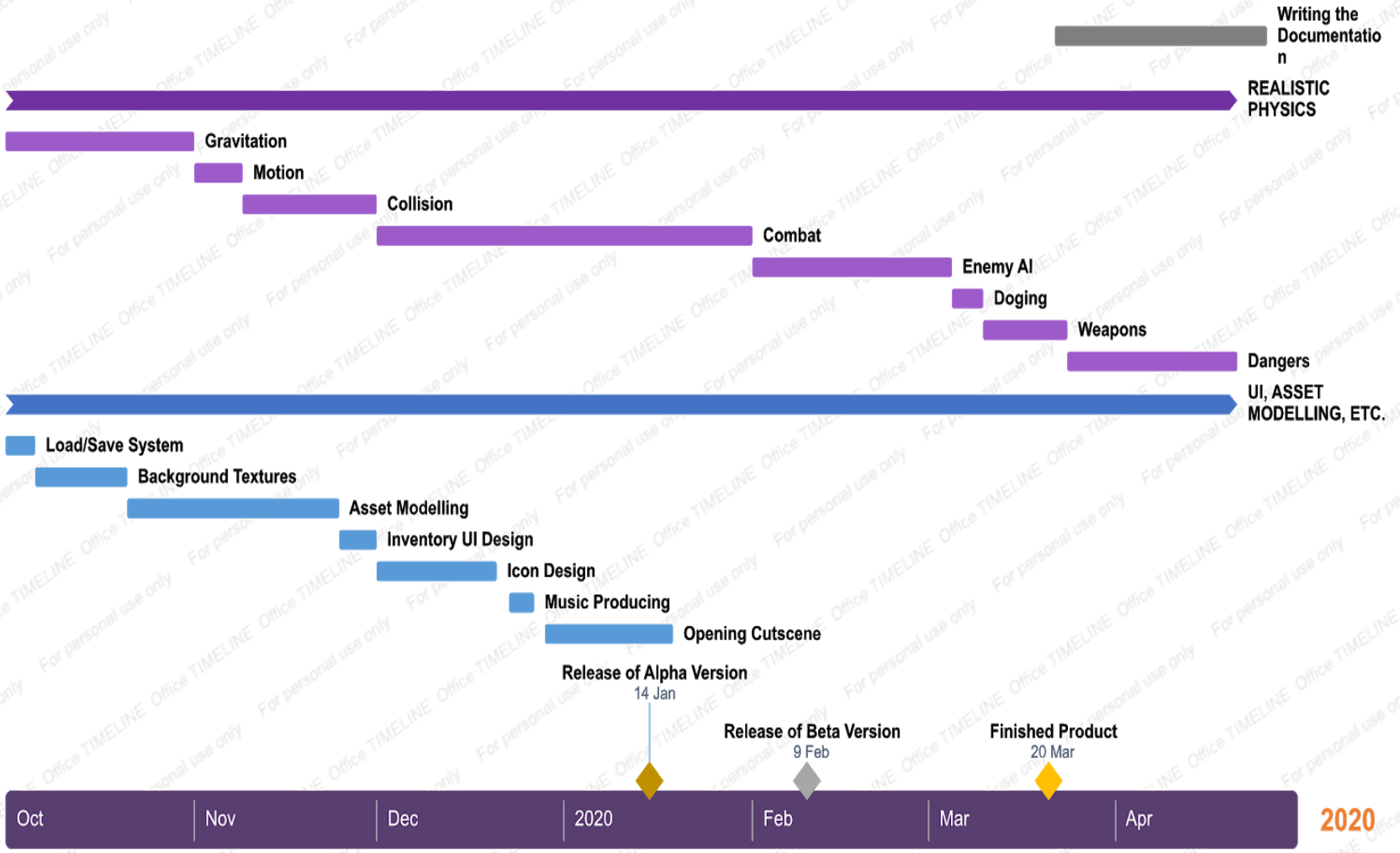
The arrows in the more saturated color represent the category of all the tasks listed below in rectangles. The length of the rectangles and arrows visualizes the estimated time needed.

Important to note is that due to the fact that one task of one team member may intersect with another task of another team member, the possibility remains that some tasks will be executed in a different chronological order than presented below.

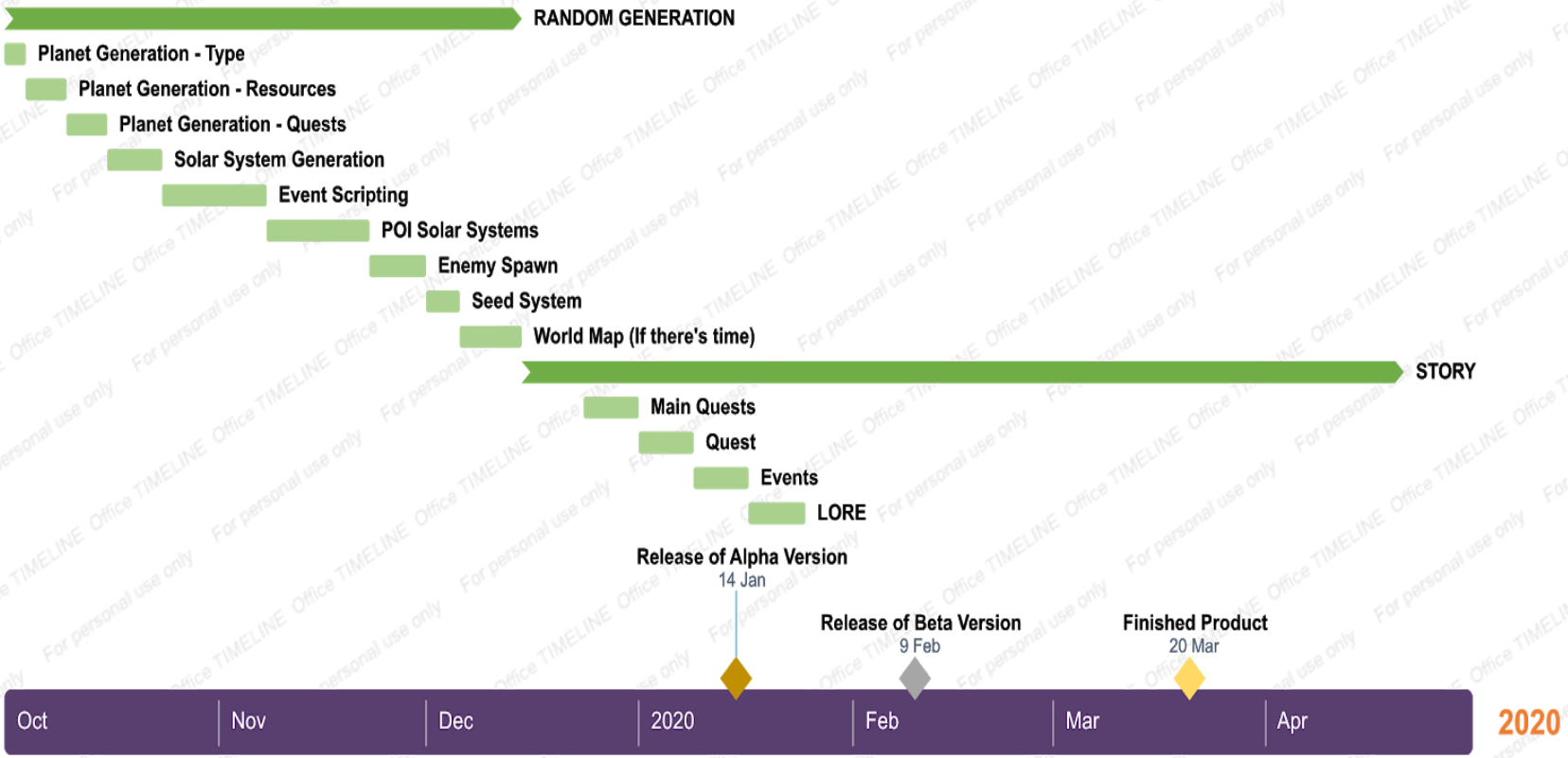
Ondrej


ITEMS
Base Class
Consumables
"Irregular Shapes"
Module Items
Crafting Resources
INVENTORY
Grid
Drag'n'Drop
Drop Highlighting
Basic Manipulation
Free Spaces
Invnetory GUI
Tooltips
CRAFTING
Tree
Recipe System
Craft Methods
SHIP MODULARITY
Ship Editing
Module Functionality
Destroyable (?)
Release of Alpha Version
14 Jan
Release of Beta Version
9 Feb
Finished Product
20 Mar
Oct
Nov
Dec
2020
Feb
Mar
2020

Vojtech and Sebastian



Denis



3.4 PLATFORMS, TECHNOLOGIES, FRAMEWORKS

The target platform for which this game is being developed is Windows.

The software and services for designing, developing and synchronization will be Unity3D, Blender, GitHub, Google Drive and Magix Music Maker.

3.4.1 Unity3D

Our choice of Unity3D as our favorite game engine is the fact that there is a huge amount of learning material, it is beginner friendly and it has a broad community which means there are numerous Unity3D users who already designed or implemented assets, scripts etc which they offer for free or for little money at the Unity Asset Store.

3.4.2 Blender

Blender is the preferred software for modelling assets and direct the opening cut scene. It is free to use and very powerful, but the main reason for the usage of Blender is the ability of quick exportation of Blender files to Unity3D.

3.4.3 Photoshop and Inkscape

Inkscape will be used for icon design mainly since the software is perfectly suitable for creating vector graphics. The main usage of Photoshop will be the creation and polishing of background textures.

3.4.4 GitHub and Google Drive

In order to save our individual progress and synchronize it, we will be using the version control system GitHub. GitHub is suitable for “light” files like scripts. “Heavy” files like assets, models etc. will be stored in a Google Drive folder, to which every team member has access to.

3.4.5 Magix Music Maker

Music Maker is a software for producing music tracks and will be used to compose the soundtrack of the game.

4. SUMMARY

The subject of this software project is the continuation of the development of an already existing game prototype with the working title “Lost In Space”.

This development consists of the “polishing” of existing features and the expansion of the game with new features. Aim of this project is the development and design of a game, that comes in terms of implementation and “look-and-feel” as close to state-of-the-art indie games as possible with only spending as little money as possible. The additional features implemented are — just to name a few — a realistic physics system, creation of AI, an intuitive user interface, several 3D models.

During the development phase the team members will have meetings to discuss their recent process and to merge the individual tasks.

The whole project is broken down into three phases — preparation phase (happening as this specification is written down), development phase which involves the release of an Alpha- and Beta-version of our game, and an end phase which consists of writing the documentation and some other post-processing.

For the team members this software project is their first “real” game development experience. The aim is for everyone to gain knowledge and experience, since our work is regarded by us as a diving board for diving into the exciting world of game development.