Project Plan

By Robin Sandvik and Viktor Chambe-Eng

Definitions

- Indie – Stands for independent. An indie game is a game developed by small groups or an individual, as opposed to a game developed by a big company.
- Deep Learning (DL) – Is a sub field of artificial intelligence that categorizes algorithms that multiple layers of learning.
- Artificial Intelligence (AI) – A broad term that covers any algorithm/procedure that simulates intelligent behavior. While often used to describe self-learning models, it also covers pre-scripted behavior like enemies in games or procedural generation algorithms.
- Procedural generation – Describes algorithms/formulas that generate output, such as a map, based on a set of rules and criteria.

1. Goals and constraints

1.1. Background

Progress Interactive AS is an independent third-party contractor for game development services. It provides 2D artists, 3D artists and programmers to larger game studios and projects.

The company has worked on several entertainment game and serious game projects, including, the hosting of the Peer Gynt Game as SaaS for schools and private distribution, a Kinect game at Hunderfosen Adventure Park, prototype apps for Mental Health Youth, and contractual artists and technical leadership for the bafta-winning My Child game. The company is also exploring the development of its own proprietary game IPs and has a small data-centre server for providing online game services. The company's funding is acquired through the Norwegian Research Council and business to business profits.

The company has been struggling with making affordable games in a market dominated by bigger companies. This problem has been made even worse by the recent pandemic.
To that end, they have tasked us with finding practical solutions for cutting down the cost and time to make games without having a significant negative impact on the product.

This project will focus on aiding Progress Interactive with a game they're currently making, but we'll also try to come up with solutions that can be used in future games. The main task they have for us is to develop a terrain generation algorithm for their game. This will act as a foundation for populating the game with structures, towns, and people, which we will also assist with once the world can be generated.

1.2. Project goals

The goals are broken down into 3 categories.

- Result goals describe what we aim to develop and make.
- Effect goals describe the effects we aim to have on the company through our work.
- Learning goals describe what we aim to learn and get experienced with.

1.2.1. Result goals

Our primary goal is to create a map generation algorithm for the game that can procedurally generate interesting maps at a reasonable performance. The resulting maps should be convertible to and from

RGBA images. Once this is functional, we will continue to improve it iteratively, as well as build other systems that work on top of the terrain generation, including systems for placing buildings and roads.

1.2.2. Effect goals

Through this project, we intend to reduce the workload and time of making indie games to help it stay affordable.

More concretely:

- Reduce the time and money needed to make indie games.
- Help indie games compete with bigger companies.

1.2.3. Learning goals

Through this project, we want to learn about game development and relevant fields. Primarily:

- Project work in realistic situations. (SCRUM, planning, adapting)
- Game design and game development.
- Procedural generation and algorithms.
- Unity and C#.

1.3. Constraints

- The map generation algorithm must be compatible with Unity/C#.

2. Scope

2.1. Field of study

Manually making each map limits the number of maps and takes a lot of work hours on top of usually having to have specialized design staff. Hiring specialized staff is very costly for small companies since they can't divide their time on different projects.

Additionally, whenever changes are made to balance or something is added/removed, the maps need to be redesigned. With a flexible algorithm, the new resources/biomes/structures/mechanics/etc. can be added to the algorithm or parameters tweaked to balance them out, then new maps can be generated rapidly. It also allows for more experimental development, where the developer can make changes, then quickly test them with a new map, then adjust and repeat.

This project will touch on many fields, including:

- Procedural generation
- Game design
- Unity/C#

2.2. Boundaries

The primary focus of this thesis is development, not research. Though we will be researching other bachelors and projects for inspiration and understanding, this is ultimately to improve the quality of the code we develop. We are also not expected to finish the game. We are only focused on certain systems within the game, and it is unlikely to launch right away after we are finished.

2.3. Task description

The primary task of the thesis is to develop the terrain generation algorithm. We will also develop additional mechanics and features on top of this system, such as a placement system for buildings and roads, as well as a basic combat system, if time permits. This will require research into procedural generation and balancing.

The terrain algorithm needs to be deterministic, meaning that given the same inputs, it gets the same output. It also needs to be able to generate more terrain on demand, such that we can generate new areas on the map days or weeks after making it.

The described algorithm must be configurable, with many parameters to allow a varied pool of maps to be made. We must be able to update it with new rules and assets with relatively little work, to integrate future changes. The employer also requested comprehensive documentation on how the system and code works, as well as how to use and modify it.

3. Project organization

3.1. Responsibilities and roles

- Group leader - Viktor Chambe-Eng
- Developers - Robin Sandvik, Viktor Chambe-Eng
- Advisor – Mariusz Nowostawski, NTNU Gjøvik
- Employer – Richard Barlow

3.2. Routines and group rules

- Each group member is expected to work an average of 30 hours per week.
- We have decided to meet physically on campus 5 days a week, from 9:00 to 15:00. The schedule can be changed at the start of a sprint if we all agree, as long as total hours remain the same.
- We have a log for the number of hours worked each day by each member, sorted by topics like coding, research, documentation, and planning.
- We will have weekly sprint meetings with the employer every Monday, as well as weekly meetings with the advisor to discuss our progress. This can be changed later if meetings seem too frequent, as long as we decide on a new regular schedule.
- All meetings must be documented, with date, time, duration, and topic.
- In case of disagreements or conflicts within the group, we can consult the employer for implementation issues and the advisor for other issues, such as routine and workflow. If there is still disagreement after this, and the employer and advisor have no opinion on the matter, the group leader must make the decision. This seems unlikely however, as we have worked together before and have had minimal conflicts.

4. Planning, follow-ups, and documentation

4.1. Choice of development model

We chose the agile development model, with weekly sprints. We will have a scrum meeting at the start of each week where we make the sprint backlog. We have made a product backlog with a rough idea of the features we want to implement, but this represents a flexible scope.

We chose this development model because the scope is so flexible. It is difficult to know in advance how much we will be able to achieve in each area. Our focus will be on creating a minimum viable product, with each aspect being functional as fast as possible. We will then add features and polish based on how much time is available.

4.2. Use of development model

We chose to go with a 1 week sprint duration from Monday to Friday. Due to the unpredictability of the project, being able to reorient ourselves often will be an important factor. Using Monday to Friday not only makes it easier to organize sprints, but also gives us the weekend to clear our head after reviewing the sprint before planning the next one. We may change the sprint duration to 2 weeks if individual issues become big enough, and don't require as much communication with the employer.

Before each sprint, we will have a quick meeting to pick what tasks we want to focus on for the new sprint. After each sprint, we will have a short meeting to review our progress and update our backlog. Due to scheduling issues, we may not always be able to have this review meeting with the employer at the end of each week, in which case we will instead send a short report digitally.

Due to our small group size, short sprint durations and how we plan to work together physically as much as possible, we opted to not do daily standup meetings and instead talk directly when needed.

5. Organization of quality assurance

5.1. Organization

Code structure

We will focus on keeping the code easy to expand and iterate over, as well as easy to tweak if we decide to change certain mechanics. This will be done by using a template method design pattern, where various parts of a larger process can be swapped out without negatively affecting the other parts. For instance, changing how the altitude map is generated should not cause the biome generation to break, though they may still affect each other in some ways, for instance a biome spreading along a valley. We will also utilize object-oriented programming and namespaces, to separate tasks and keep the code both easily expandable and changeable.

Comments and documentation

The code must be well commented, to keep it readable both for us and others working with the company who may use our code in the future. We will have a Doxygen-inspired style of comments from the start for functions, as well as shorter comments for more complicated or important code. Once the employer begins reviewing code, they may request additional documentation, in which case we will expand our existing comments to ensure it remains readable not only to us, but to anyone who may work with our code.

Testing

We will constantly be testing the terrain generation as we add and tweak features. To make this more efficient, we will generate many maps simultaneously with different seeds. This will give us a bigger picture of what the terrain generation is capable of and help expose edge cases and unlikely scenarios. Later in the development process, when the terrain generation becomes more defined and we focus on details and balancing, we will need more extensive testing. This will require more creative testing methods and will be a significant challenge later in the project. We might for instance try to calculate what the maximum possible size of a biome is, and generate many worlds as fast as possible and measure if this is accurate.

5.2. Configuration management

We will lock the main branch, forcing all commits to happen on our own branches that we will then merge into the main branch. We will attempt to keep our commits small, but might occasionally have bigger commits, for example when we need to make many or large changes at the same time for code to work.

5.3. Tools

- Sharepoint – Cloud storage of documents
- Microsoft Word – Writing documents
- Jira – Sprint planning and backlog
- BitBucket – Source code repository
- Discord – Communication and meetings
- Unity – Making code for the game
- VSC/VS – Code editors

Sharepoint

We will use the Sharepoint service provided to us by the university to store our documents on the cloud. This integrates well with Microsoft Office, which we will be using to write documents such as the reports, time logs, and meeting logs.

Jira

We will be using Jira to plan our sprints and BitBucket for repositories. This was requested by the employer.

Discord

We will be using a discord server we set up for communication, sharing resources and conducting meetings. This will be used by the group, employer and advisor.

Visual Studios Code/Visual Studios

We will use VSC/VS as our code editor. This provides us with a lot of feedback and information about our code while coding, helping us avoid errors/mistakes.

5.4. Risk analysis

Below is a list of risks regarding the project. They are each assigned a low, medium, or high value in likelihood and consequence.

1. Too big a scope. The scope could turn out to be too much for us to develop in the limited time we have. Likelihood: medium. Consequence: medium.
2. Group member becomes unavailable. Due to illness or other unforeseeable events, a group member could be unable to work on the project for a time. This would be especially impactful, as there are only two group members. Likelihood: low. Consequence: high.
3. Poor communication between group and employer. Since the employer has a limited schedule, we might sometimes have to rely on messages rather than meetings if we have any questions, or they have requirements. This could lead to misunderstandings in the short term. Likelihood: high. Consequence: low.
4. Internal conflict within group. We could have disagreements about how to implement certain features or how to manage work, to the point where it reduces our efficiency. This should not be very likely, as we have worked together before and have chosen a group leader, as well as having advisors to consult. Likelihood: low. Consequence: medium.
5. Loss of work/code. Losing a device or access to a server could result in large amounts of work being lost. Likelihood: low. Consequence: high.

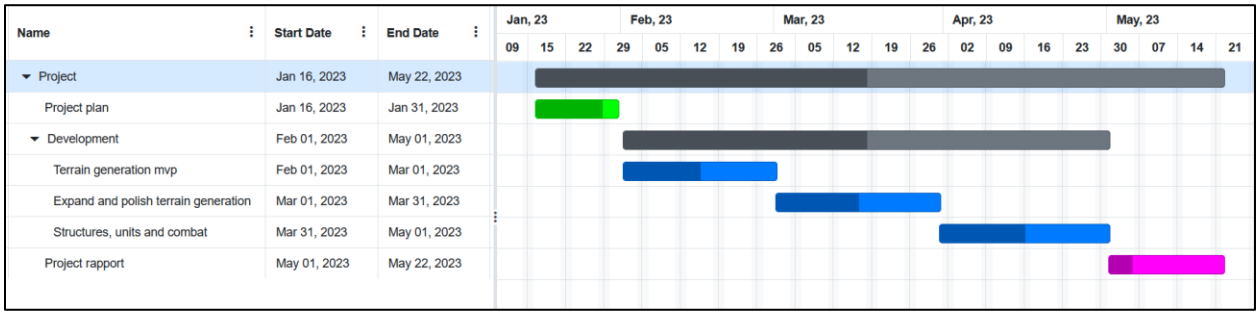
	Low likelihood	Medium likelihood	High likelihood
Low consequence			3
Medium consequence	4	1	
High consequence	2, 5		

From this analysis we have identified the most serious risks. Below are the proposed countermeasures to limit the likelihood or consequence of these risks.

#	Risk	Countermeasures
1	Too big a scope	Use agile development method and work on issues iteratively, to ensure working functionality as fast as possible. This reduces both consequence and risk.
2	Group member becomes unavailable	Work closely and communicate often about the work we do. Use thorough documentation and comments. Do not separate tasks too much, so one member can take over where the other left. This reduces consequence. Little can be done to reduce likelihood.
3	Poor communication between group and employer	Have regular sprint meetings and discuss thoroughly what the specifications of each issue are. Have a separate message channel for important questions when meetings are not possible. This reduces likelihood, as well as consequence, as misunderstandings would be resolved quickly.
5	Loss of work/code	Use a repository rather than keeping code locally on a device, and commit frequently. Additionally, use trusted platforms like bitbucket. This reduces both likelihood and consequence.

6. Development plan

6.1. Gant-diagram



6.2. Milestones

- Project plan: 31/01/2023
    - Deliver project plan.
- Terrain generation MVP: 01/03/2023
    - Have a fully working MVP of our terrain generation model.
- Expand and polish terrain generation: 31/03/2023
    - Expand and improve the MVP iteratively.
- Structures, units and combat: 01/05/2023
    - Integrate structures and units to the model, and work on combat if time allows.
- Project rapport: 22/05/2023
    - Deliver the final report.