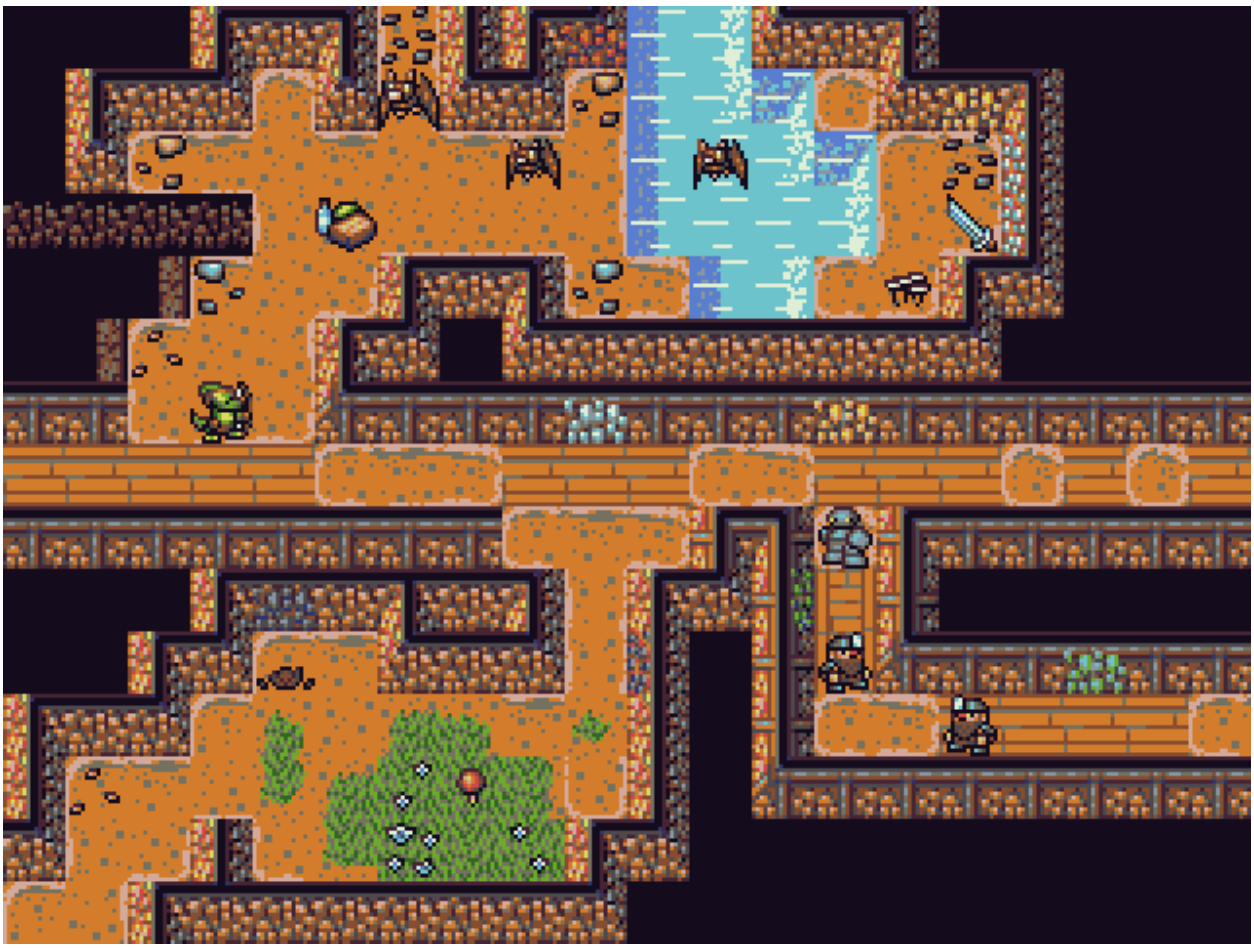
Andrew Meckling
A00854442

COMP 8045 Project Proposal Document

Developing a tile-based dungeon crawler using component-oriented programming in C++.

Rynth



Rynth (as in labyrinth)

Rynth is a 2D, tile-based, D&D inspired, dungeon crawler with two interesting twists.

Innovation One (Puzzle Mechanics)

Firstly, the dungeons serve two purposes: at a small scale (squares of 9-25 or so tiles) the dungeon layout provides the combat area for players and enemies, at a larger scale (squares of 25-225 or so tiles) it provides the puzzle state of the dungeon. To put it another way, when zoomed in the game is about combat, when zoomed out the game is about solving puzzles. (Smaller scale puzzles will also be included as a means of solving parts of the larger scale puzzle. These are referred to as micro- and macro-puzzles. More on this later.) These large-scale puzzles will take the same basic form as the classic puzzle game *8-Puzzle* (or *15-Puzzle* depending on the size of the board). In 8-Puzzle, the board is a 3 by 3 grid of tiles with one tile missing so that the other tiles can be freely moved around and rearranged by the player (8 is the number of non-missing tiles). Generally, the tiles will have numbers on them but some versions have images on the tiles so that when the puzzle is solved the entire board appears as one picture.

Figure 1. A children's 8-Puzzle toy using a picture instead of numbers.

In Rynth, these tiles are represented by large squares of game tiles. (Author's note: due to the confusing double usage of the word *tile*, I will attempt to clear up some possible confusion here. Unless otherwise noted, "tile" refers to a *game tile*: a 1 by 1 square which can either be a floor, a wall or a pit. Regions of these types of tiles make up the layout of the dungeon and can hold up to one player, enemy, item, etc. on them. In most cases, "room" refers to a large square of game tiles which represents a single *puzzle tile*: a region in the dungeon of fixed dimensions which can be moved in any of 4 directions provided it will not overlap with any other puzzle tiles or the edge of the board. Puzzle tiles may be fixed in place or have their movement restricted to a set path.) Instead of trying to reach a single specific state for the puzzle, i.e. the solved state with the picture completely unscrambled, the intention of the player with regards to the puzzle tiles is to arrange the rooms to form a path connecting previously inaccessible areas. Because there will not necessarily be a picture to help guide the player to a solved state, the puzzles will generally not be scrambled very severely and could have multiple possible solutions (this may change after play testing). During the loading phase of each level the dungeon will first be instantiated in the solved state and then scrambled using only valid puzzle tile movements. These movements will be recorded and used to give the player hints if they are stuck for too long. This circumvents the need to implement an 8-Puzzle solver which can be quite slow for puzzles with more than 15 tiles: $O(b^d)$ for breadth-first search and $O(b^m)$ for depth-first search where b is the branching factor, d is the depth of the solution and m is the maximum tree depth. It is possible that more than a single puzzle tile will be missing from some of the dungeons if difficulty proves to be an issue. The idea is not to recreate a faithful implementation of 8-Puzzle but to provide an interesting game mechanic that requires the player to think across small and large scales.

Innovation Two (Death Mechanics)

Secondly, Rynth attempts to innovate on the profoundly unexamined yet still ubiquitous *death mechanic* found in many games. Death provides a straightforward and easy to understand fail state. You died. Try again. Usually tied to a health mechanic, traditional death mechanics are unreasonably effective at indicating to the player how well they are performing at any given moment of gameplay (the health mechanic is only meaningful because it triggers the death mechanic). The issue is that nearly every game has some sort of death mechanic and they are all more or less the same. It has become an artifact of game design which forms an understandable language across innumerable games from dozens of genres. One benefit to this is that the death mechanic rarely must be taught to new players. The most successful innovation on death mechanics, and there aren't many to choose from, is *permadeath*: death doesn't mean "try again," it means "start over." From the beginning. Popularized by *Rogue: Exploring the Dungeons of Doom* (or just *Rogue*), permanent death has been featured in many games since its release in 1980, spawning a whole sub-genre of games, which are like Rogue in several other important aspects, called roguelikes. In addition to permadeath, Roguelikes also generally feature dungeon crawling, procedural generation and role-playing mechanics; all of which, except for permadeath, have seen countless variations and innovations between games. Rynth aims to fix this issue.

In Rynth, death is never the end. Dying in Rynth is something the player may or may not want to avoid depending on the situation. When a player "dies," they are resurrected in an alternate version of the dungeon they died in. This alternate version may have a somewhat dissimilar layout and different enemy and item placements. Each dungeon will have at least one alternate version, or *dimension*, to visit. Subsequent deaths will cycle the player through other or previous dimensions. This way of treating death impacts the game in two important ways. 1. enemies defeated in one dimension may show up in another, ready to fight you all over again if you happen across them during your interdimensional travels. 2. some micro-puzzles can only be solved by traveling to other dimensions, i.e. dying. Macro-puzzles will have the same configuration between dimensions. Each dimension will have its own unique set of enemies. Enemies from different dimensions may be hostile towards each other. One issue that arises from this unique take on death is that it doesn't provide a fail state. However, this may prove to be less of an issue than it first seems; puzzles don't often need a fail state. This issue will be alleviated at least partially by including a cost to death, such as consumption of a resource. This resource will likely be an experience or gold penalty (experience being the means of making the player character stronger and gold being a means of acquiring better items or weapons), plus some sparse resource thematically linked to resurrection and interdimensional travel (for now these will be referred to as *death tokens*). The player may be able to spend their death tokens to travel between dimensions with a reduced or eliminated penalty cost. If the player dies without any death tokens it will trigger a game over.

Core Mechanics

Rynth takes its primary inspiration for combat from *Dungeons & Dragons*. Characters have a health pool and a maximum amount of health; when their health reaches 0 or less they "die." Combat takes the form of rounds. Each round, player and enemy characters take their turn. A turn consists of movement and actions. Movement is straightforward and will be explained in full detail later. If movement is the bread, then actions are the butter. Both movement and actions will deplete a *time meter* which indicates how fast the character is at performing their actions/movement. Different actions will consume different amounts of the time meter. A character will be able to take any action if they have at least some amount

of time left in their meter. When the time meter is fully depleted the turn ends. Actions will be things such as attacking, using character specific abilities (e.g. setting traps), casting spells, using items, interacting with the environment (e.g. pulling a lever) and so on. Defeating enemies will yield experience points to the player which can be used to strengthen their character. Killing the same enemy again in other dimensions will yield less experience each time. Enemies may also drop loot when defeated. Loot can be anything from gold coins to health potions to weapons and armour. Enemies can only drop loot the first time they die.

Implementation Details

Rynth will be programmed by myself in C++ using a proprietary engine of my own design—the engine itself is nameless but will be dubbed the *Rynth Engine* for the sake of documentation. The core of the game will be built on an Entity Component System (ECS). Early stages of development will use OpenGL for rendering but will be reimplemented using the Vulkan API once the majority of the gameplay systems have been developed. Audio and music will be controlled via FMOD. Interfacing with the OS (such as receiving keyboard input and mouse position) will be handled through SDL.

Engine

The Rynth Engine is proprietary software which I have been working on for the last two years during my studies at BCIT. The engine is by no means a complete package. Work on the engine has been and will continue to be done as features are needed by this project and others. The features of the engine are designed to promote loose coupling and limit OOP techniques to cases which necessitate them (such as the scene management system which takes inspiration from Android's *Activities*). Care is taken during development to make sure features added to the engine are general enough to be used by any game and aren't specific to the game being developed with the engine at the time.

Entity Component System

Primordial versions of the Rynth Engine were based on a naïve entity component framework implementation. As I learned more about ECSs and how they have been implemented in industry I revised and improved my approach. The current version makes use of variadic templates, bitsets, arrays, tuples, forwarding references, lambdas/function objects and hash maps. The following is a mock implementation of the ECS used in the Rynth Engine:

```
struct Entity
{
    int      index;  // Position in manager.
    unsigned bitset; // Indicates which components are attached.
};

template< size_t Size, typename... Components >
class ComponentManager
{
    static constexpr size_t SIZE = Size;

    template< typename T >
    using Storage = std::array< T, SIZE >;

    using OccupancyStorage = std::bitset< SIZE >;
    using EntityStorage = Storage< Entity >;
    using ComponentStorage = std::tuple< Storage< Components >... >;
```

```
    OccupancyStorage occupancy;
    EntityStorage    entities;
    ComponentStorage components;

public:

    // Gets the specified component of an entity.
    template< typename Component >
    decltype(auto) get( Entity& ntt )
    {
        return std::get< Storage< Component > >( components )[ ntt.index ];
    }

    // Attaches a component to an entity.
    template< typename Component >
    void attach( Entity& ntt, Component&& component )
    {
        get< Component >( ntt ) = std::forward< Component >( component );
    }
};
```

Figure 2. A highly simplified ECS implementation. Note that the tracking of occupancy and attached components is not depicted.

Rendering

Since its inception, rendering in the Rynth Engine has been done through open OpenGL via the programmable graphics pipeline. Going forward, the Rynth Engine will use the Vulkan API for rendering. The Rynth Engine has some facilities to create primitive shapes on the GPU but most of them will not be needed due to the tiled nature of the graphics in Rynth (more on this later). Rynth will be rendered using 2D sprite-sheet based shaders. Because everything in Rynth is square shaped (transparent pixels notwithstanding) advanced techniques such as vertex buffers and vertex arrays are largely unnecessary. That said, they will still be used in conjunction with the programmable graphics pipeline. A single square-shaped vertex buffer will be stored on the GPU and used for all sprite rendering.

Audio and Music

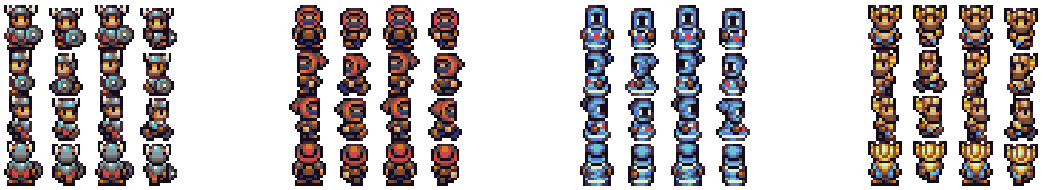
The Rynth Engine received its first real audio implementation about halfway through its life in the form of FMOD, an industry standard in terms of audio mixing and tool support. FMOD can dynamically add or remove components of a sound track based on what is happening in the game. For example, in a stealth game the audio can play louder or more frantic once the player has been discovered and gradually revert to normal as the enemies lose sight of the player. All of this is controlled by a single variable passed into the FMOD implementation which adjusts the sound track in a way which was configured using the FMOD studio tools. FMOD also supports randomizing the pitch of audio on each playback which is particularly useful for repetitive sounds like footsteps. It also supports 3D and 2D audio.

Keyboard and Mouse

Input from human interface devices (HIDs) is handled via a lightweight abstraction on SDL's event system. Each frame the entire input state is copied to a buffer and the previous state is moved to a different buffer. This makes it trivial to determine in code whether a particular key or button was pressed or released on the current frame or simply being held down. The Rynth Engine also supports controller input from any number of controllers.

Scope and Depth

Rynth is a single player game, inspired by *Dungeons & Dragons*. The game will feature three stages each with multiple levels to complete. The first stage will be designed to teach players the game mechanics and won't include the macro puzzles described earlier. The very first level will not include the interdimensional travel death mechanic but instead will be used to set up its inclusion in the game thematically. Art assets for the game have been procured from opengameart.org (https://goo.gl/Pg8Dzd) and include environment, decorations, enemies, items and four player classes (warrior, rogue, mage and paladin).



Player Classes

Each class will have different base stats, access to different spells and abilities plus a single gameplay altering ability, e.g. the warrior will build a "rage meter" when taking damage which unlocks special effects on some of his abilities and the paladin will be able to smite some enemies, preventing their resurrection in other dimensions. These abilities are not finalized and are likely to change and evolve over the course of development as more features are implemented and tested.

Base Stats

The base stats in the game are as follows:

- Hit Points (HP): Represents how much damage a player can take before dying (and being transported to another dimension).
- Attack (ATK): Represents how much damage a player will deal with their attacks.
- Magick (MGK): Represents how much damage a player will deal with abilities and how often they can use their abilities.
- Speed (SPD): Represents how many actions a player can take on their turn.

The base stats can be increased as the player levels up by spending experience points which are gained by slaying enemies in the dungeons.

Rough estimate of stat break down per class (each row sums to 16):

	Hit Points	**Attack**	**Magick**	**Speed**
Warrior	**7**	4	2	3
Rogue	5	4	2	**5**
Mage	5	3	**5**	3
Paladin	6	4	3	3

(At the time of writing, the paladin class is meant to be unlocked after the first successful playthrough of the game, but this may change during development.) Each enemy will also have a value for each of these stats.

Abilities

Players will have access to up to four abilities at any one time. Abilities will be selected from a pool of abilities available to each class before entering a dungeon. This will create a "loadout" metagame because players will have to decide which abilities they want and which ones they don't. Abilities are activated by pressing a button indicating which ability to use followed by another button indicating which direction to use the ability in. Some examples of abilities:

Warrior:

- Crescent Slash: Deals damage to enemies in a 'C' shape in front, to the sides and diagonally adjacent to the player.
- Shield Block: Dramatically reduces damage taken from a direction by enemies until your next turn.

Rogue:

- Hide in Plain Sight: The player is invisible to enemies who are not already in combat with the player until the player moves or performs an action.
- Set Trap: Sets a trap adjacent to the player that will damage the first enemy to move on top of it.

Mage:

- Cone of Cold: Deals frost damage to the enemy in front of the player and any enemies behind or diagonally adjacent (in the same direction) to the enemy.
- Wall of Fire: Ignites the ground in a line in front of the player for the next 2 turns, dealing fire damage to anyone caught in the flames.

Paladin:

- Holy Smite: Deals damage to an enemy in front of the player. If this kills it, it will not be resurrected in another dimension.
- Lay on Hands: Fully heal either the player or a character adjacent to the player.

Levels and Stages

Rynth is broken down into three stages, each with a set of levels designed with a particular intent in mind. The first stage will feature levels designed to get the player accustomed to the dimension traversing death mechanic as well as the basic combat and movement mechanics. The second stage is designed to expose the player to the macro-puzzles. The final stage is designed to test the player's skills at combat and ability to think across dimensions at a large scale (the entire dungeon). The first two stages will have only two dimensions per level but the third stage may have as many as three. Each stage will have approximately five levels for a total level count of 15. Levels will vary in size and enemy types (which will also differ across dimensions).

Art Assets and Enemies

Assets for several hundred enemies, each with an additional frame for animation, are included in the assets procured from opengameart.org (https://goo.gl/Pg8Dzd). There are really more assets than I could ever possibly use which is great as it puts a high ceiling on my creative freedom for making Rynth. Each level will add a new type of enemy or new micro-puzzle type. The target for number of enemy types is 30.

Most enemies will only have a single attack but will have different stats. The large number of enemies is intended to provide aesthetic variety rather than gameplay variety, per se.

Micro-Puzzles

Micro-puzzles are intended to add gameplay variety without affecting the combat gameplay. Rynth will have 3-4 types of micro-puzzles. (For the rest of this section they will be referred to simply as puzzles.) The puzzle types are as follows:

- Lights Out: Pressing buttons will toggle one or more lights. The goal is to turn all the lights on or off.
- Sliding Blocks: Blocks will slide in the direction they are pushed until colliding with an obstacle. The goal is to either push a block through a maze or to arrange blocks so as to open up new paths for the player or other blocks.
- Invisible Walls: Walls in a room are only visible when standing right next to them (otherwise they look like floor). The goal is to solve a maze without being able to look ahead very far.
- Rectangle Escape (Rush Hour): Rectangles can be moved along one axis but cannot overlap with other rectangles or obstacles. The goal is to move a particular rectangle out of the field of obstacles.

These puzzles will see several configurations throughout the levels of Rynth. Completing one of these puzzles will either open a door, reveal a lever which is used for moving puzzle tiles (as mentioned in the puzzle mechanics section above) or give the player an item necessary to progress to the next level or area of the dungeon.

Milestones, Deliverables and Testing

Rynth, in its entirety, will be completed by November of 2017. The completed project, an executable that will run on Windows, the source code and the final report will be ready by the end of November. Before that happens there are several major milestones which will need to be achieved.

Major Milestone 1 (Dungeon Editor and Basic Gameplay System)

After the first month of development (June) Rynth will be in a playable state. This means that after launching the game the player will be presented with a test dungeon and dialog to choose which class to play as. The player will be able to move about the dungeon and attack enemies which will attack them back. At this point the game will declare the player a winner once all enemies have been defeated. If the player dies they will be resurrected at the start of the dungeon (without affecting their progress). Dungeons will be created by launching the game with the "-edit" command line option. The editor will present the user with an empty dungeon (filled only with floor tiles). The user will be able to choose a tile type (wall, floor, pit) to paint onto the dungeon surface with the mouse. The user will also be able to place enemies, items and start location in the dungeon using a drag and drop interface. Dungeons can be saved to file with Ctrl+S. Dungeons will be saved in a quasi human readable text format. Dungeons can be loaded with Ctrl+O.

Major Milestone 2 (Character Class Progression and Level Progression)

After the second month of development (July) stage 1 of the game will be mostly completed (including micro-puzzles). This means that when the game is launched the player will be able to choose which class

to play as and will be put into the first level of the game. Upon completing each level, the player will be taken to the next level. When there are no more levels, the player will be declared a winner. The player character will gain experience (EXP) from defeating enemies and at certain EXP milestones their base stats will all be increased by 1. Each level will have two dimensions which the player can switch between by dying (except for the first level). Players will have access to at least 2 abilities per class which will be integrated into the combat system. Player stats and abilities will be displayed with text at the side of the screen.

Major Milestone 3 (Macro-Puzzles and Loadout Customization)

After the third month of development (August) stage 2 of the game will be partially completed. The macro-puzzle system (sliding rooms around) will be completed. Tentative layout of macro-puzzles will be established but will undergo revision after playtesting. After selecting which class to play as the player will be able to choose up to 4 abilities (from a pool of at least 4) to bring with them during their entire play session. Some enemies will have abilities they can use against the player during combat. Stage 1 of the game will be finalized. The game will include a main menu which will allow the user to pick a level to play (for now) or load a level from file to edit.

Major Milestone 4 (Interface Improvements and Vulkan API)

After the fourth month of development (September) stage 2 of the game will be completed and stage 3 will be partially completed. The main menu, pause menu and HUD will be finalized. The pause menu will include “resume”, “restart”, “settings” and “quit” options. The main menu will include “new game”, “continue game”, “load game”, “settings” and “quit” options. Not all menu options will be fully implemented at this time. The OpenGL rendering implementation will be switched over to Vulkan; the engine is designed in such a way that this won’t require the modification of any game code, only engine code. By this point the final report will be well underway and will receive continuous attention throughout the remaining development.

Major Milestone 5 (Items, Abilities and Balance)

After the fifth month of development (October) stage 3 of the game will be completed. All abilities and items will be implemented into the game. All menu items will be fully functional. Rigorous playtesting, tweaking and balancing will be documented.

Major Milestone 6 (Polish and Stability)

After the sixth month of development (November) the entire game will be complete and polished. The game will be stress tested and crash tested (end process).

Testing

Testing will be carried out continuously during development. Functional requirement testing will be performed and documented regularly during each day. User testing will be performed on a weekly basis after the first milestone. User tests will consist of friends and family members playing the game. Users will first be briefed on the changes made since their last test before testing begins. After testing, users will be asked about their thoughts on the gameplay and mechanics as well as the level of polish. Functional requirements will be documented in the following format:

Test Name	Test/Feature Description	Testing Actions	Pass Condition	Fail Condition	Outcome
Player Movement	Players can move around the dungeon and are blocked by walls, pits and enemies	Pressing the arrow keys to move the player around. Attempt to move through walls, pits and enemies	The player moves in the intended direction and does not move off of floor tiles	The player either moves in the wrong direction or moves through walls, pits or enemies	
Enemy Pathfinding	Enemies will follow the player once they are within “aggro” range, avoiding walls, pits, other enemies and hazards (such as fire or traps)	Move the player near an enemy	The enemy follows the player through the dungeon and avoids hazards and impassable terrain	The enemy gets stuck on corners, doesn’t avoid hazards or fails to follow the player at all	

Expectations and Technical Challenges

Rynth will be the sixth game in a row I have worked on for BCIT (the first being my diploma program final project). I spend a large portion of my free time programming in C++ so I more or less feel like I know what to expect from this project. Rynth will be the first game I’ve worked on (and completed) all on my own, which I perceive to be a major advantage. There’s nothing I enjoy more than game development and I expect to treat my work on this project as a full-time job (30-40 hours per week). I see only three challenges in this project:

1. Dungeon saving/loading
2. Vulkan API
3. Effective level design with puzzle-tiles and unique death mechanic

The first challenge is mitigated by the fact that I have written two level editors for games in the past, one of which was also a tile-based game. In the event that dungeon saving/loading is too much to manage in a reasonable amount of time I will fall back to hard coding the levels with text file input only for the layout (determining which tiles are floors, walls or pits). The second challenge should be reduced as I work my way through more Vulkan tutorials (at time of writing I am only on the third chapter of the official introduction to Vulkan site). I have a real passion for computer graphics and efficiency therein—I fell in love with OpenGL almost as soon as we began learning it; my first implementation of an OpenGL renderer was in 3D with .obj material support. The third challenge is the one I am most eager to tackle. This is the second game in which I will have to design and test puzzles. I plan to use physical 8-Puzzle toys to help give me an intuition on how to design these puzzles. I will also be looking at puzzle levels in games I have played to help me design the micro-puzzles for Rynth (such as the gyms and caves from the Pokémon games). I will be taking inspiration from the famous *Effect and Cause* level from *Titanfall 2’s* single player campaign to design levels for the dimension shifting death mechanic.

Summary

Rynth is a 2D, tile-based, D&D inspired, dungeon crawler programmed using component-oriented techniques in a custom-built game engine with C++. Rynth innovates on similar games in two interesting ways. Firstly, dungeons are not just hallways and rooms, they are puzzles; entire sections of the dungeon can be slid past each other to reshape the dungeon and open new paths while closing old ones. Secondly, death is never the end. Whenever a character—be it player or enemy—dies, they will be resurrected in an alternate dimension with subtle differences that are key to making it through the dungeon. Players can take the role of several classes, warrior; rogue; mage or paladin, each with a unique set of abilities and their own strengths and weaknesses. The game is broken up into three stages, each with approximately five levels, designed to introduce the concepts of the game to the player gradually. Combat and movement follow a simplified *Dungeons & Dragons* formula. Art assets for Rynth have already been selected and supply more art than could ever be used in one six-month solo project. Rynth will be the first game developed using the Vulkan API support in what is now dubbed the "Rynth Engine."

About the Author

Andrew has been interested in games and modding from the age of 13, finally deciding to take the Games Development program at BCIT when he was 15. In high school, he took every computer course that was available to him: 3 years of web development and one year of programming in visual basic. Andrew has completed the CST program at BCIT. Math, in particular physics and geometry, has always come easy to him and lead to his interest in computer graphics. In his spare time, Andrew studies techniques for writing efficient code that is both cache friendly and easy to use. He specializes in writing game engines and systems from scratch in C++, a language which he excels at. Were he not developing a game he would undoubtedly resume work on developing a programming language which takes the best features from C++ and makes them easier to use in the context of game development.