

Roadmap for Dunder Dig:

**Game Engine:**

Most important step! Need a collection of functional, modular game – engine- things in order to streamline development, testing and understanding scope and limitations. The goal here is to make completely modular, self-contained game objects that can be instanced during game development, rather than engineering things on the spot. Example, making a dedicated text engine to service text bubbles in-game. Make a dedicated text-bubble object that can be instanced. This step is the prep stage of a big recipe. It’s vital to have all the vegies chopped and meat seasoned so when it’s time to cook, I can just start tossing things in. This step also serves as an opportunity to not only brainstorm many ideas for the future of its development, but to begin ensuring a strong compatibility between the scope of all these ideas, and the scope of the project. Pending list of game objects to develop for this step:

Player Character: movement, gravity, speed, etc. Abilities, duck-tales’ thrust.

Text Bubble: Specifically, the physical bubble. It’s size, position, animations, large character faces in the corner, pointy thing to show who’s talking. How it will implement the text-engine.

Text-Engine: Ok so for this thing, I want to be able to very easily change the font, text size, color, and maybe even place images in the chats. A simple solution for this is each character in a string to be printed will be a ‘Stamp’ object, which can either hold one character, a string of text, or an image. A virtual typewriter.

Stamp Object: Holds either a character, string, or image. Contains attributes to font, size, color, image size, etc. Maybe implements some sort of linked-list scheme so each stamp contains a pointer to its previous or successor, in order for implicit placement when typing long strings of text. Also, static functions to simply return a linked list of stamp objects from an inputted string of text. The whole idea of this thing is to abstract the idea of text so that it would be as easy to include images as it is it include text. In addition to this, it will also be much easier to change the color, font, and size of specific characters/ lines of text. Idk it sounded good the first time I thought about it.

UI: User Interface for the player in-game. To include idk. Money, health, xp, map something like that. The goal for this step is to simply implement any functional UI elements and make it as modular as possible in order to be able to add/subtract elements in the future.

Inventory System: When the user presses a key, a self-contained inventory menu will appear, and take focus from the input. The inventory is ‘owned’ by the Game since there is only one player.

Player Character: Put this one again because this is actually probably the biggest step, and the most important to get right. I think the best policy is to tinker and toil until the player movement is as enjoyable, intuitive, and satisfying as possible. This is truly the backbone of the entire game. It’s the most frequent input, and the largest interface between the player and the software. No one will want to play the game if its not fun to move the character. Idea here is to implement a duck-tales / shovel knight movement , jumping, and that down-pogo needs to be just right. There is no shame in trying to emulate shovel knight’s as much as possible since they nailed it perfectly.

Each power up the player will get, will enhance their standard move set and be re-instanced as a separate game object. Example character move sets:

Duck Tales/Shovel Knight Pogo, Homerun Bat from Smash Bros, Shoot-em-up spaceship controls, <list is pending>

Signs: players can read signs for hints. They either will just trigger a text bubble or will just be an NPC so that I don’t have to implement a real sign object. It’s not like having separate objects is particularly cumbersome, but it might be funny to blur that line between a static sign and a dynamic NPC.

NPCs: These also just trigger a word bubble, but they have flags for how many times you talked to them, animated sprites, uhhh.

Wall: A wall has collision, can be placed in any orientation (a floor is a ‘wall’.) Tile based.

Dirt: A dirt is a wall that you can break (with the shovel).

Collectable: When the player walks into a collectable, it disappears and the game is updated (flags, UI, inventory, etc.).

Enemy: Has collision, will damage the player. Can be damaged by the player. Maybe its also just an NPC (that can be collided with and damaged. )

Projectile: It moves in a fixed path , can be parried or pogo-d on. Will damage the player.

Shop System: Like the inventory system, when triggered, will bring up a newly focused menu.

[**2 weeks later]**

Main Story: This is being worked on, thought about seamlessly with the other components, but by the end of this phase, ideally the entire story should be finished.

Plan Levels: This goes a little in tandem with the story planning but should have all the levels mapped out at this point to include NPC interactions and enemy placements. At the beginning of this phase, all the ‘objects’ to create a level should already exist and be ready to toss in. During this phase, I will make rough sketches of the levels, and then simply instance those already-done game objects, and graphics-tiles. At the end of this phase, the levels will be effectively done (with place holder / alpha art).

Finalize Game engine: The game engine is just a collection of modular parts and will never be truly finished. At the end of this stage its crucial that there is a clear direction and scope of the entire project, and it should also be clear what needs to be added or fixed from the first stage. Example “ oh we planned to use a mana-bar so now I need to update the UI. Or, we decided recently to have a bomb-level, so I need to add a bomb- object to the game engine. “

**[4 weeks later aaa]**

The game is done! Now I need to add music and art! (I do not know how to make music and art). In order to efficiently update game art, keep the directories nice a neat and I think simply replace place-holder art with different file of the same name, and Godot will automatically import it correctly. Polish might be the single most important step! This encompasses the smaller details, shading, dynamic lighting, sound effects, more jokes, alternate dialogue, and fine-tuning level design. Exporting the project will certainly not be a simple task. Might take days or weeks of refactoring, googling, and fighting Godot to finally get a Windows .exe out of this damn thing. Then there is bug fixing and making it work on Linux/ mac (I do not care about doing this).