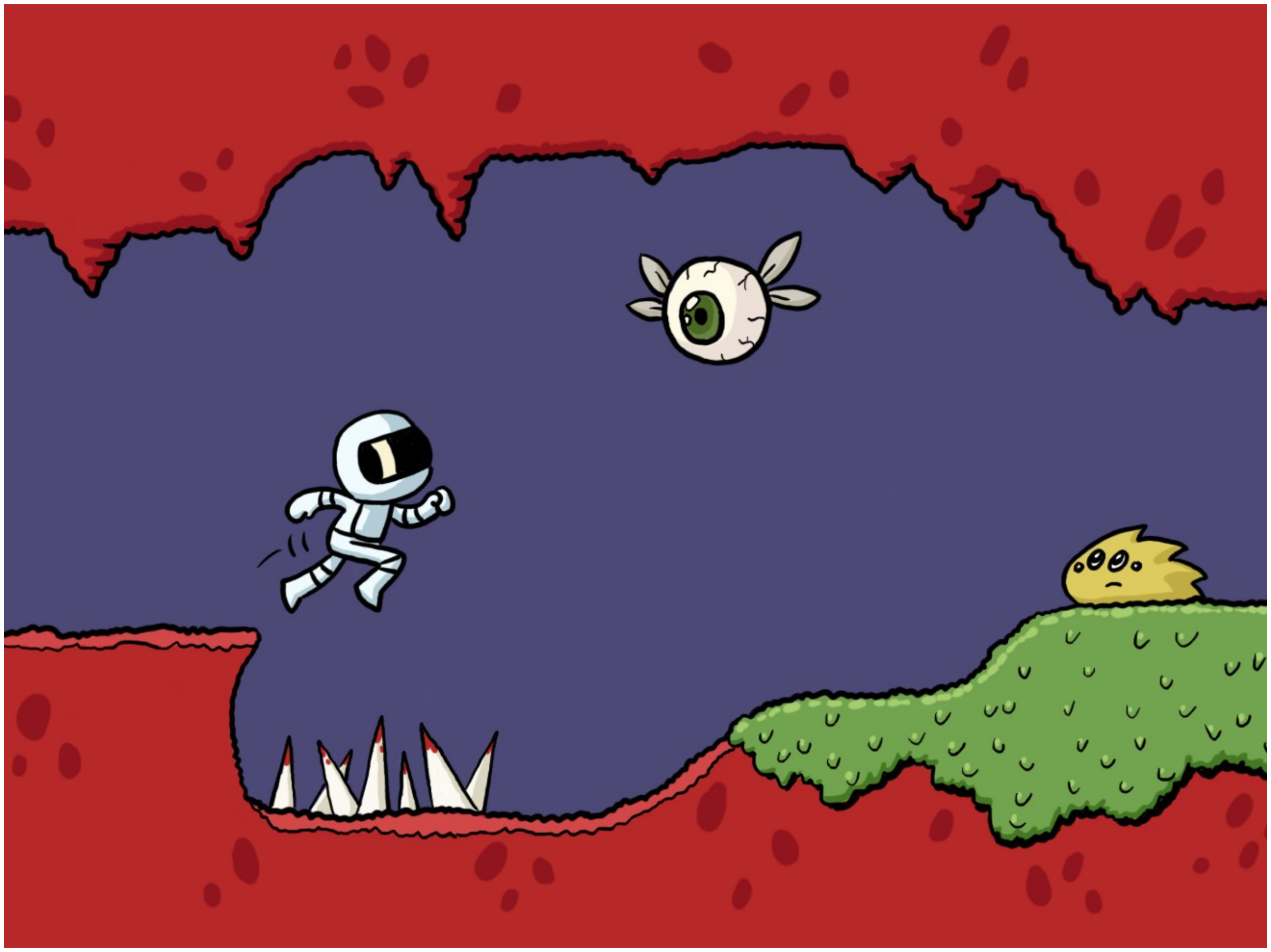
Design Document

Planetary Pitstops



Nathan Moore
October 3, 2013

Revised October 7, 2013

Section 1. Introduction

While out on an ordinary journey through space, an astronaut's personal planet hopping rocket get's damaged – causing it to lose all of the energy crystals powering it. There is only enough fuel left to land on a near by planet to try to find enough energy crystals to make it to the next planet – a little bit closer to home. Each planet will pose a new set of challenges as he eventually arrives at his destination.

This will be a side scrolling platforming action game similar in style to classics like Metroid, Megaman, and Cave Story. The primary game play mechanics will involve running, jumping, a small jet-pack boost while jumping, and shooting enemies along with, possibly, and swimming. This game will most appeal to gamers who have grown up on playing Nintendo and Super Nintendo games and will have the level of challenge expected from games of those eras.

Section 2. Overview of the Game

The game will consist of a series of action platforming levels. In each level the player will need to fight through enemies and avoid obstacles to collect all of the available energy crystals. There may be difficult jumps, aggressive enemies, and a few bosses making this a challenging task. Throughout the levels there will be EMTs (Emergency Medical Teleporters.) If the player loses all of his health, he will be teleported back to the last EMT that had previously been activated by jumping on it.

After collecting the crystals, the player will need to return to the ship and fly off to the next planet. After completing all planets, the little astronaut will be safely reunited with his home planet and the player will win the game.

There will be three save slots for the player to choose from at the beginning of the game. The game will save at the beginning of each level and at EMTs allowing players to pick up and put down the game as needed until completion.

This will be a graphical game, but the graphics will be kept fairly simple to reduce development time. There will also be music and sound effects to enhance the game play experience. No score will be kept track of, but the game may keep track of a cumulative play time for people to be able to challenge themselves by beating the game more quickly.

Section 3. The User Interface

The game will be controlled with either the keyboard or a joystick/controller. The arrow keys, joystick, or directional pad will cause the astronaut to run left and right. The keys and buttons can be configured in a configuration file, but there will be a default layout. The “S” key, “2” button on a joystick, or the “A” button on an XBox 360 controller will cause the astronaut to jump and pressing it again while in air will make a small rocket boost that propels him up a little bit higher. The “D” key, “1” button on a joystick, or the “X” button on an XBox 360 controller will cause the astronaut to fire his laser pistol.

On screen, the player will see the level, the astronaut main character, enemies, and items. At the top of the screen will be the player's health information, the number of crystals remaining to be collected, and maybe a mini-map to help with navigating caves.

The levels will be broken up into areas with each area focusing on some set of game play elements. The player can move from area to area by walking or jumping off the edge of the screen (the camera follows the player until it bumps up against an area boundary.) For example, one area may have flying enemies and platforms over lava and, after moving off the right side of the screen, the player will move into an area with a boss battle.

Most of these areas will reset after leaving and returning to them – thus replenishing dead enemies. Some areas, though, will use “accomplishment” values while initializing to respond to what the player has already previously done – for example, to prevent fighting bosses again and allowing the player to only collect each energy crystal once.

Section 4. Architecture of the Game

There will be an attempt to design the game in a way that that will provide a reasonable amount of functionality while, at the same time, remaining relatively simply. Game objects will be composed mostly of reusable components (sprites, collision objects, simple physics, animation controller, behaviors, etc.)

Some areas will have some scripting to help add some story and interesting dynamics. Much of the dynamics, though, will be achieved though "wiring up" objects by having them subscribe to publishers in other objects – for example, a door can subscribe to a button and listen for the button to send the "pulse" message that is caused by the player interacting with it. To simplify design, the standard messages will be "on" when something is turned on, "off" when something is turned off, and "pulse" when a short signal (like a button press) occurs. This will allow for different kinds of objects to easily be used together in different situations.

The game will have a debug mode that can be toggled on and off by pressing F3. This debug mode will pause the game and allow a person to click objects to see bounding boxes, state information, and other values about each object. It will also allow for quickly moving the player anywhere in the area, to load a different level, or to reload the current level to see changes. There may be some other functionality added as needed.

Enemy artificial intelligence will be fairly simplistic – like most older games. Some might move left and right and turn around at walls or edges, some might fly around randomly and maybe swoop down at the player, some will simply have repeating patterns of behavior like jumping at regular intervals.

The game will make use of several different kinds of assets. Along with PyGame images, sounds, and fonts there will be files for defining animation sequences, tile maps in Tiled's file format (likely with the comma separated values setting for simplicity), level scripts, tile properties, configuration files, save files, etc. Most of the custom file types will be in JSON format for easy loading and saving. Levels will be defined almost entirely in the Tiled tile map editor.

Section 5. Scope of Effort

This game will likely be a little bit larger than it should be. Because of this, not all the features may be implemented by the deadline – for example, there may not be very many planets to explore and not many enemies or bosses. It may be that only a game play prototype (with player controls and simple interactions) will be achieved within the allotted time frame.

There will be modules and classes for the game application, game states, scenes, asset management, animation, messaging, game objects, object components, levels, tile maps, scripting, tracking accomplishments, saving and loading progress, visual effects, configuration, testing and debugging, player inputs, user interface elements (like the mini-map, health bar, and dialog boxes), and a few more.

The amount of content that goes into the game depends largely on the amount of time is remaining before the deadline. There may only be two or three enemy types, or, if time permits, more than that. Likewise, the number of planets depends on the time needed to construct them.

Section 6. Implementation Plan and Timeline

The first version of the game will be a skeleton framework implementing the most important features. This will include loading stripped down versions of the levels (no scripting), some of the player controls (no swimming), a dumb test enemy (just a blob), shooting, getting hurt, and player dying. There will be one test level which is designed mainly to test game play functionality like tile map collision and object interactions – this level will be neither fun nor pretty.

The second version will add more enemy types, a boss or two, a more well designed level, and more engine features. Some of these engine features will handle more advanced levels with scripting, moving between areas, componentized game objects, visual effects, simple gui widgets (like dialog boxes for displaying instructions and story), messaging between objects, and maybe a few other things.

The third version will finish up and fill out any missing parts along with adding more game content. More bosses and enemies will be added along with more levels. Game play and graphics will also be polished to be a little more enjoyable. A simple story will be added, more or better music, and better sound effects.

Time will be an issue, since progress on this needs to be made around other assignments, each with their own deadlines. This is also a lot of work for the small amount of time available to do it in. The first version of the game, itself, may need around thirty hours of development time[1], with each subsequent version requiring twenty or thirty more hours. Finding a hundred or more hours of free time to develop this would be a real challenge.

Section 7. Resources Used

The plan is to use the following applications and tools during development:

- Python (http://www.python.org).
- PyGame (http://www.pygame.org).
- The Gimp (http://www.gimp.org/) for graphics.
- Tiled (http://www.mapeditor.org/) for designing tile maps.
- Audacity (http://audacity.sourceforge.net/) for sound editing.
- Bfxr (http://www.bfxr.net/) for generating sound effects.
- LMMS (http://lmms.sourceforge.net/) for composing music.
- GitHub (https://github.com/) for version control.
- PyCharm 3 Community Edition (http://www.jetbrains.com/pycharm/) as an IDE.
- Notepad++ (http://notepad-plus-plus.org/) for general text editing.
- Font Squirrel (http://www.fontsquirrel.com/) for finding a font to use.

Section 8. Concluding Remarks

From time to time, some of my older projects might be referenced to more quickly

1 Estimate based on a similar sized PyGame project that I've previously made over the course of one weekend. It was a pretty intense two and a half days, but tile maps, level loading, level scripting, enemies, sound effects, music, platforming game play, message boxes, animation, and simple enemy behaviors were implemented in around that amount of time. See http://www.youtube.com/watch?v=eTrgNZyjhjU for a time-lapse video making it.

implement features. There isn't any plan to copy and paste any of the source code, though, because much of it can use refining and redesigning. This might help shave off some development time, but maybe not too much.

The project's GitHub page is https://github.com/ArmchairArmada/COS125Project01/.

Section 9. Bibliography

[1] Curtis Meadow (2013) “COS 125 Design Document”

Section 10. Work Assistance Statement

All of the programming will be done by be, but I plan on having a few people play test the game and give feedback on it. Some of their ideas might get added into the game. My brother, or a friend of mine, may want to contribute in small ways like by drawing some graphics or designing a level or two. This, generally, is supposed to be a solo project, but since they are not helping with any of the programming it might not be an issue.

This design document and all it's contents were made by me while referencing the design document template we were given.