Team Rigel
Tucker Walker
Marisa Rea
David Pipitone

<Adventure Game>: Final Report

15th March 2019

OVERVIEW

<Adventure Game> is a video game adventure that pulls inspiration from many different games such as Zelda, Pokemon, and Mario Bros. It is written in C++ and playable in a Linux environment. The user is able to control a playable character while navigating a 2-D world via a top-down or side-scroller view. Interactive windows allow the user to switch between world navigation and inventory. Windows are also used to display important stats such as health and the currently equipped item, as well as display important narrative information such as the room descriptions, item descriptions, or events that occur.

USER PERSPECTIVE

<Adventure Game> at its core, is a video game. As such the user can move around and interact with objects within any of the given rooms. The game will be reminiscent to the user of vintage video games, both in the visual design and the tribute to various vintage games in the sixteen puzzle rooms which comprise the complete game map.

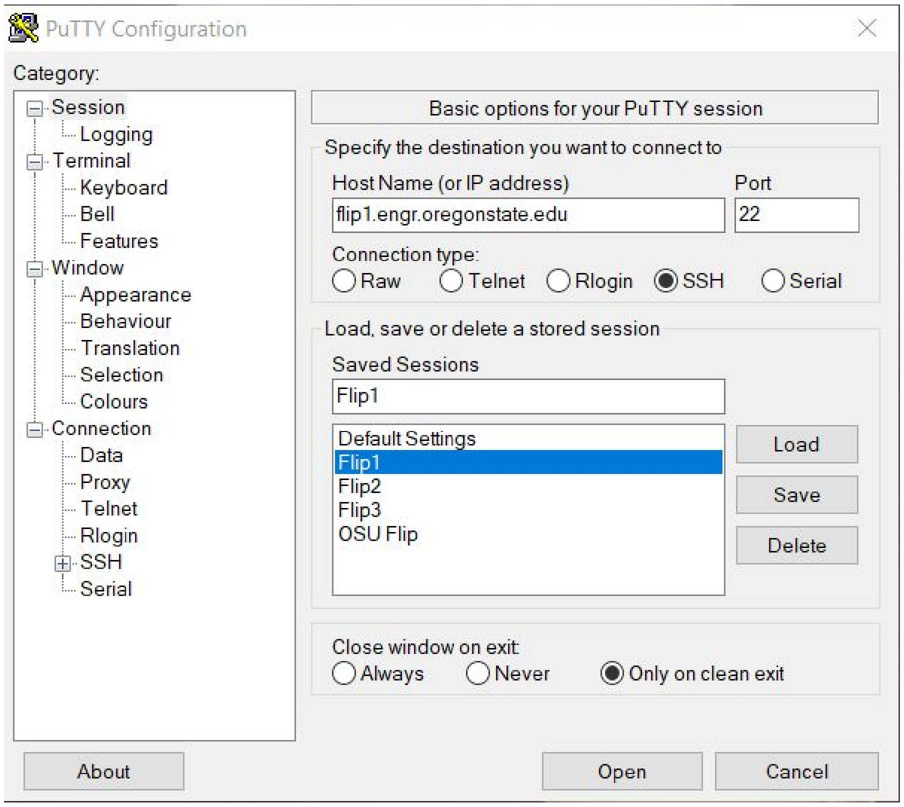
In a world of high tech graphics, the classic look and feel of <Adventure Game> will aid in the nostalgic experience that the game aims to provide, facilitating the immersion of the user into the vibrant memories of their gaming past. The top-down Roguelike game display and mechanics will feel intuitive to the target user, with the surprise side-scrolling control adjustment in the final puzzle rooms surprising and reinvesting the player into the gameplay. The change in POV and game mechanics will cause the user to pause and remember they are playing not just a video game or a school project, but interacting in a nostalgic exhibit of our cherished childhood memories as simple nerds and geeks, eager to get in to the world we would later know as “Computer Science”.

HOW TO COMPILE AND HOW TO PLAY

The following instructions will go over how to run the game as well as the controls. There will also be a brief walkthrough of <Adventure Game>. However, if further detail is required for the walkthrough or items that will be encountered, it is strongly encouraged to view the Game Guide pdf that is included with the submission. We kept the Game Guide separate from this report because it was a milestone to create a standalone guide and we felt it would detract from the overall value of the report by including everything that was found within the Game Guide.

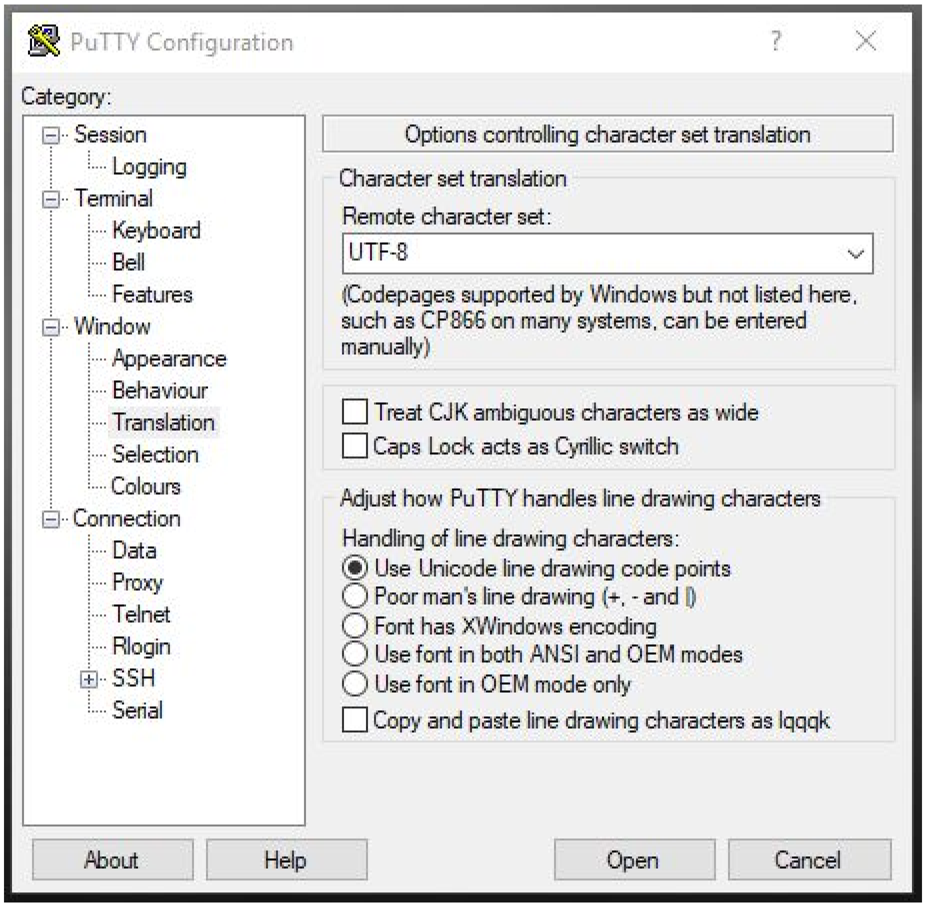
Running The Game

1. Set up your Terminal with PuTTY
    a. Open PuTTY
    b. set **hostname** to *flip1.engr.oregonstate.edu* and **port** to *22*



Host Name (or IP address): flip1.engr.oregonstate.edu
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Saved Sessions: Flip1
Default Settings
Flip1
Flip2
Flip3
OSU Flip
Close window on exit: Always Never [✓ Only on clean exit]

c. Select **Window->Translation** and set **Remote character set** to *UTF-8*



Remote character set: UTF-8

(Codepages supported by Windows but not listed here, such as CP866 on many systems, can be entered manually)

- [ ] Treat CJK ambiguous characters as wide
- [ ] Caps Lock acts as Cyrillic switch

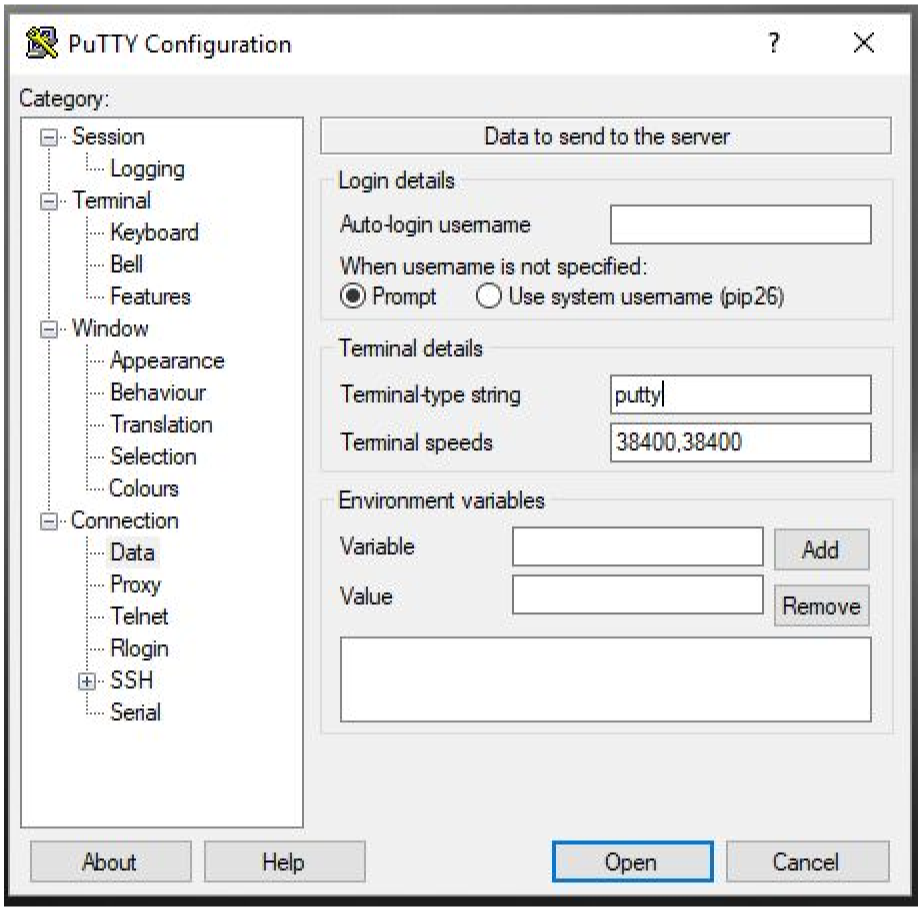
Adjust how PuTTY handles line drawing characters

Handling of line drawing characters:

- ● Use Unicode line drawing code points
- ○ Poor man's line drawing (+, - and |)
- ○ Font has XWindows encoding
- ○ Use font in both ANSI and OEM modes
- ○ Use font in OEM mode only

- [ ] Copy and paste line drawing characters as lqqqk

d. Set *Connection->Data->terminal-type* string to **putty**



PuTTY Configuration

Category:

Data to send to the server

Login details

Auto-login username

When username is not specified:
(● Prompt) ○ Use system username (pip26)

Terminal details

Terminal-type string: putty

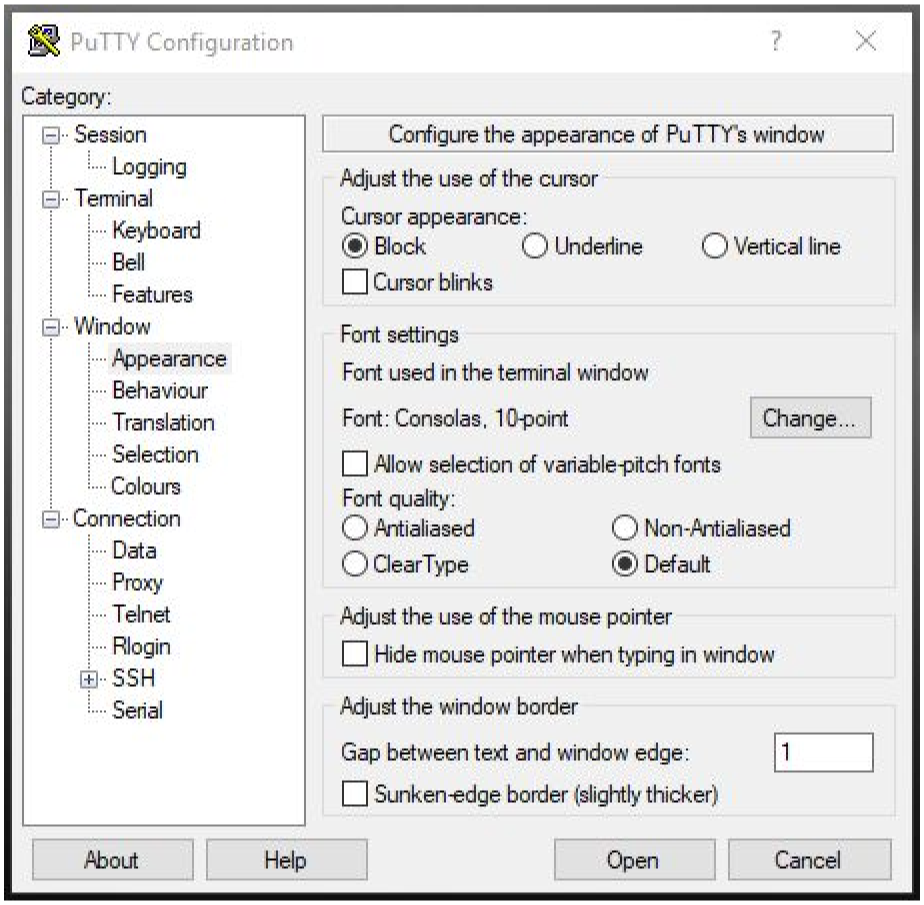
Terminal speeds: 38400,38400

Environment variables

Variable

Value

e. Set to **consolas** to putty's font



Configure the appearance of PuTTY's window

Adjust the use of the cursor

Cursor appearance: [✓ Block] Underline Vertical line

☐ Cursor blinks

Font settings

Font used in the terminal window

Font: Consolas, 10-point

☐ Allow selection of variable-pitch fonts

Font quality: Antialiased Non-Antialiased ClearType [✓ Default]

Adjust the use of the mouse pointer

☐ Hide mouse pointer when typing in window

Adjust the window border

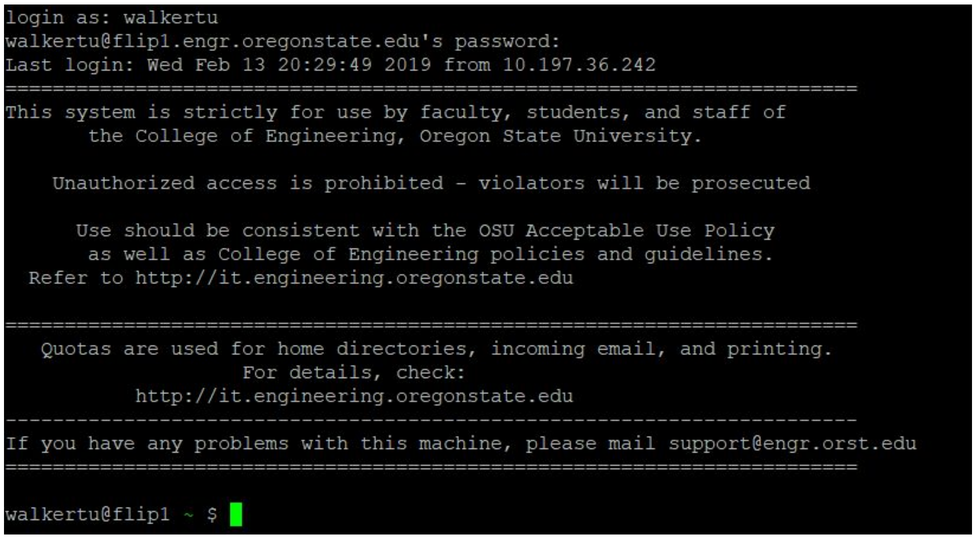
Gap between text and window edge: 1

☐ Sunken-edge border (slightly thicker)

f. Select **Open**

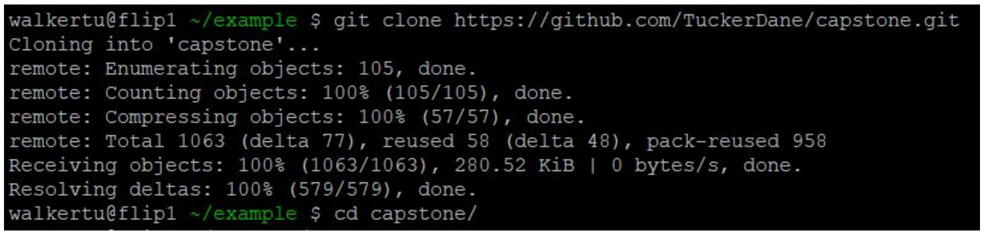
g. Maximize the terminal window

h. Log in



```
login as: walkertu
walkertu@flip1.engr.oregonstate.edu's password:
Last login: Wed Feb 13 20:29:49 2019 from 10.197.36.242
========================================================================
This system is strictly for use by faculty, students, and staff of
       the College of Engineering, Oregon State University.

    Unauthorized access is prohibited - violators will be prosecuted

      Use should be consistent with the OSU Acceptable Use Policy
       as well as College of Engineering policies and guidelines.
  Refer to http://it.engineering.oregonstate.edu

========================================================================
   Quotas are used for home directories, incoming email, and printing.
                    For details, check:
           http://it.engineering.oregonstate.edu
------------------------------------------------------------------------
If you have any problems with this machine, please mail support@engr.orst.edu
========================================================================

walkertu@flip1 ~ $
```

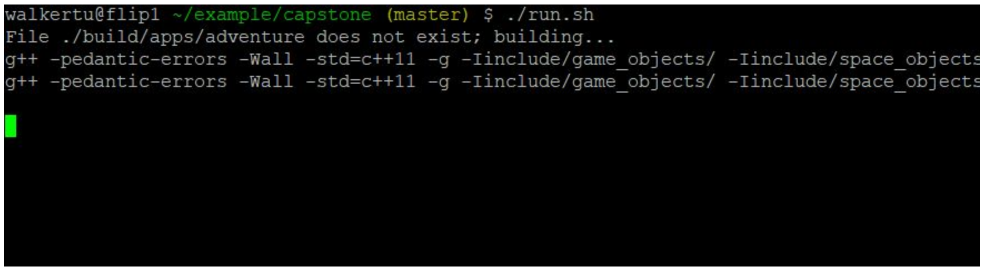
2. Download **<Adventure_Game>**
    a. Navigate to the directory that you want to run **<Adventure_Game>** inside of
    b. Type *git clone https://github.com/TuckerDane/capstone.git* and navigate into the **capstone** repository with *cd capstone*



```
walkertu@flip1 ~/example $ git clone https://github.com/TuckerDane/capstone.git
Cloning into 'capstone'...
remote: Enumerating objects: 105, done.
remote: Counting objects: 100% (105/105), done.
remote: Compressing objects: 100% (57/57), done.
remote: Total 1063 (delta 77), reused 58 (delta 48), pack-reused 958
Receiving objects: 100% (1063/1063), 280.52 KiB | 0 bytes/s, done.
Resolving deltas: 100% (579/579), done.
walkertu@flip1 ~/example $ cd capstone/
```

    c. If **<Adventure_Game>** is downloaded as a zip file…
        i. Unzip the file
        ii. Run *dos2unix <path_to_capstone_repo>/capstone/run.sh*
        iii. Run *chmod +x <path_to_capstone_repo>/capstone/run.sh*

3. Run **<Adventure_Game>**
    a. Type *./run.sh* to begin building and playing **<Adventure_Game>**

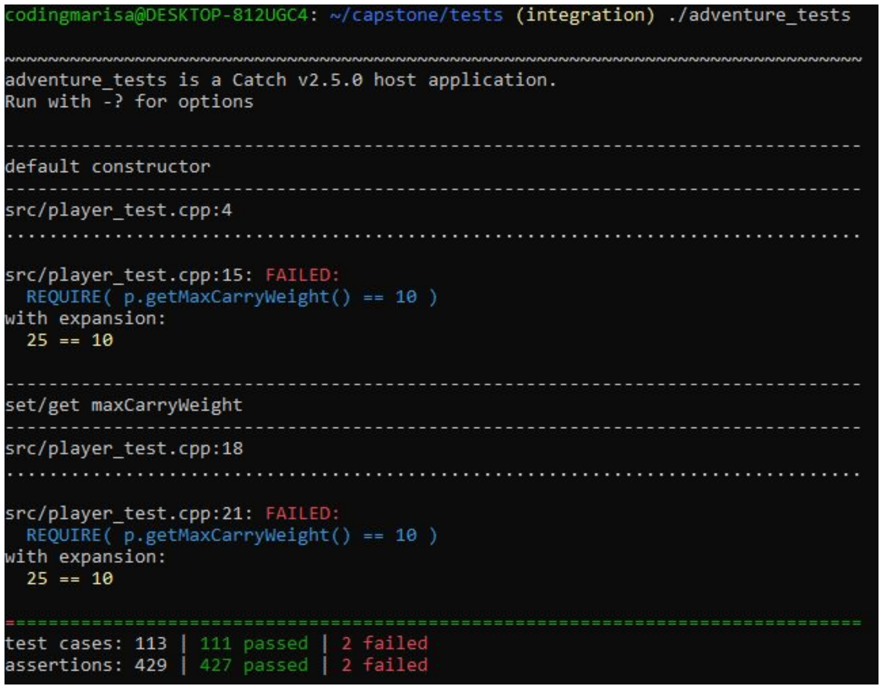


```
walkertu@flip1 ~/example/capstone (master) $ ./run.sh
File ./build/apps/adventure does not exist; building...
g++ -pedantic-errors -Wall -std=c++11 -g -Iinclude/game_objects/ -Iinclude/space_objects
g++ -pedantic-errors -Wall -std=c++11 -g -Iinclude/game_objects/ -Iinclude/space_objects
```

<u>Running the Test Suite</u>

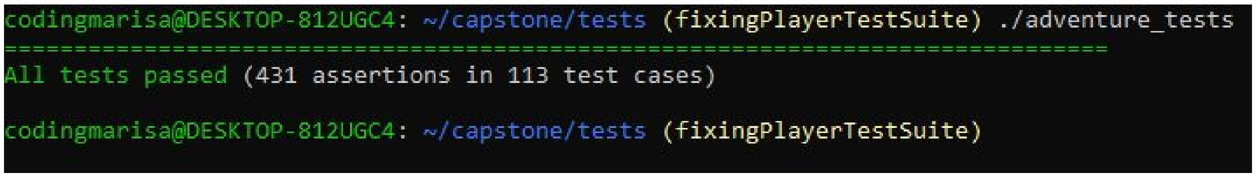
1. Complete steps 1 and 2 in the “Running the Game” section
2. Execute the command “cd tests” to navigate to the test folder
3. Execute the command “make all” to compile the test suite
4. To run the test suite, you may either run all tests, or only tests with specific tags
    a. To run all tests, enter the command “./adventure_tests”
    b. To run a test on a specific tag, enter the command “./adventure_tests [TAG_NAME]” and input the name of the tag between the square brackets

5. Tests with failures will appear as seen below:



```
codingmarisa@DESKTOP-812UGC4: ~/capstone/tests (integration) ./adventure_tests

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
adventure_tests is a Catch v2.5.0 host application.
Run with -? for options

-------------------------------------------------------------------------------
default constructor
-------------------------------------------------------------------------------
src/player_test.cpp:4
...............................................................................

src/player_test.cpp:15: FAILED:
  REQUIRE( p.getMaxCarryWeight() == 10 )
with expansion:
  25 == 10

-------------------------------------------------------------------------------
set/get maxCarryWeight
-------------------------------------------------------------------------------
src/player_test.cpp:18
...............................................................................

src/player_test.cpp:21: FAILED:
  REQUIRE( p.getMaxCarryWeight() == 10 )
with expansion:
  25 == 10

===============================================================================
test cases: 113 | 111 passed | 2 failed
assertions: 429 | 427 passed | 2 failed
```

6. Tests without failures will appear as seen below:



```
codingmarisa@DESKTOP-812UGC4: ~/capstone/tests (fixingPlayerTestSuite) ./adventure_tests
===============================================================================
All tests passed (431 assertions in 113 test cases)

codingmarisa@DESKTOP-812UGC4: ~/capstone/tests (fixingPlayerTestSuite)
```

Game Controls

All controls are key based and not case sensitive. Thus, if you want to move up ‘W‘ and ‘w’ are both valid inputs. For dropping an item both the semicolon and colon are valid inputs.

Move Up - W, up arrow
Move Down - S, down arrow
Move Left - A, left arrow
Move Right - D, right arrow
Pick Up - P
Drop - ; or **:**
Use Equipped Item - E
Opening Inventory - I

Highlight Inventory Item (Up) - W
Highlight Inventory Item (Up) - S

Equip Highlighted Item - E (while in the Inventory Window)
Quit Game - Q (Prompt will appear. **Y** to quit, any other key will not quit)
Access the DevConsole - ~ or `

Brief Walkthrough

A more detailed walkthrough and more information about items encountered in <Adventure Game> can be found in the Game Guide.

The Player starts in the starting room. This room was designed to be a tutorial room. The player collects a few keys that open the doors in this room and must advance through the green door. A Cube Piece is located in the room and must be collected to successfully complete the game.

The second room consists of a block puzzle. The blocks must be moved and both keys must be obtained. The yellow key is required to leave the room to the next room. The magenta key is required for the third room.

The third room consists of two movable blocks. One damages the Player and the other heals the Player. This room tests the Player's patience. The red block must be moved to the far right to get access to the red door. The middle section can be accessed by the magenta key. The Player should obtain both the cyan key and the Cube Piece. To progress to the next room, the player must unlock the red door with one of the keys they have previously obtained.

The next room contains a Snorlax and a Pokéball. The snorlax can be moved after obtaining a Pokéflute. Proceed south to the next room.

This is the Saffron Gym room. It contains teleporters. By using the teleporters you can access 4 different rooms, collect a potion, the Pokéflute, and the remaining 3 Cube Pieces. After collecting all the items, you can proceed back to the Snorlax room and use the Pokéflute on Snorlax. You can choose to catch Snorlax by using a Pokéball on him. After you Move Snorlax, you can proceed West to the Bomberman room.

In the Bomberman room you can use the bomb to blow up softwalls. The goal is to reach the teleporter in the bottom left.

The teleporter leads you to the Trap room. One trap in every column does medium damage, one does minimal damage, the other does no damage. The goal is to navigate through the traps without dying. The player is unable to advance through the locked white door, but they can use the teleporter.

The teleporter takes the Player to the Rodent's Revenge room. There is a key that can be collected to return to prior rooms by giving access to the Trap room. To progress the game, the Player must use the bottom doorway on the right. The top right leads back to the Trap Room.
The goal of Super Mario Bros is to reach the end of the level. To do so, the Player must complete this section to reach the second section. Avoid falling down the holes in the ground and watch out for the gravity mechanic.

The second section of the Super Mario bros is completed when the Player touches the flag. Just like Mario completing the level and moving on to the next, the flag takes the player to the next room.

Upon being teleported by the flag, the Player ends up in the End Room. If the player has caught Snorlax, then the Player can trade Professor Oak. A secret room can be accessed from this room. The teleporter is hidden. To complete the game, the Player must read (press R) while facing the Repair Station (statue in the middle of the room). Doing so will combine all 5 pieces. Having less than 5 pieces in your inventory will result in failing the game.

SOFTWARE AND SYSTEMS

The team used a combination of **PuTTY**, **Ubuntu**, and a project-specific set-up within **Visual Studio Code** to develop the code for <Adventure Game>. Visual Studio Code was adjusted to be linked directly with our local git repositories for the project. Additionally, the terminal within VSC was set to Ubuntu, as well as the Windows Ubuntu application installed on the team member computers for easy, Linux based testing during development. Further testing and development was done on FLIP with development taking place both within **Vim** and **Notepad++**.

TOOLS AND LIBRARIES

<Adventure Game> was written in **C++11** and compiled in **Linux** using **g++** and a makefile. Specifically, it is capable of compiling and running on one of OSU's three flip servers. It relies heavily on the ***ncurses* library** in order to render graphical data to the user, taking advantage of

various built-in functionalities such as *Windows*, *Colors*, and *Cursor Manipulation*. A test suite was developed using **Catch2**, an automated test case framework build using C++. Version control and coordination between each team member's code implementations was handled with **Git** via **GitHub**.

TEAM CONTRIBUTIONS

Tucker Walker

Tucker is the creator of the original <Adventure Game> demo and de facto leader of the team. He created the premise behind <Adventure Game> by dreaming up a final project inspired by classic rogue-like games but with additional features/enhancements by way of the ncurses library. After the group formed, Tucker's drive and enthusiasm for the project was channeled into early progress in the development of <Adventure Game>'s base game engine, paving the way for David and Marisa to follow-up with all of the frills and fun bits. His primary focus was set toward developing a lower-level game API. Tucker also used his current/previous work experience to help teach, inspire, and encourage the group to continuously work in a positive and productive manner. In the beginning, Tucker wrote the Project Proposal, set up the project in GitHub, and helped teammates set-up their programming environments. He created the initial structure of the game's windows, spaces, objects, and player movement. He also set-up the Catch test framework the team utilized to test new objects through the game's development. Further into the term, he created an enhancement which allowed the usage of alternative character sets to give the game a more complex and enjoyable experience. He also contributed to the development of three final puzzle rooms, the creation of alternative object types to enhance Marisa's Rodent Revenge room, as well as added additional documentation to the Game Guide. Finally, Tucker masterminded the creation of our enemy class and the relevant AI which allows the enemy characters to move independent of the player.

David Pipitone

David was the star-eyed game aficionado working behind the scenes. He used his creativity to develop the plot for <Adventure Game>, as well as brainstormed up a hurricane of great ideas for the further development that would be enjoyable – or irresistibly frustrating – for the user. Early in the term he expanded the Player/Object/World interaction, solved bugs related to player movement between rooms as well as created an alternative means of room travel, and contributed to Game Guide documentation. Further in <Adventure Game>'s development, David proved to be the team's most prolific room developer, contributing nine of the sixteen final rooms, all of which complex. He also created an alternative movement system which allowed the user to switch from top-down to side-scrolling game play in the Mario rooms as well as additional object types to be used in the game. David created the Project Poster and contributed to the Final Report and the final version of the Game Guide.

Marisa Rea

Marisa was the feature creature of the team. She worked on enhancing the base game by developing and/or building upon the functionality of the non-world window features such as the inventory window, narrative window, as well as the start and end screens. Within the world window, she looked to create rooms with object utility and functionality different than the rest. Early in the term Marisa kept the team on track through the initial development schedule, added ncurses components to the inventory window and developer console, as well as added enhancements to existing features such as the quit functionality and method of setting color. She also prepared the Midpoint Project Check and prepared weekly meeting outlines. Later in the term, she created alternative objects, expanded the test suite, enhanced narrative functionality, and contributing to the final Game Guide and Final Report. She also developed four out of the sixteen final rooms - notably, the Bomberman room which includes the time-based bomberman bomb and softblock classes. These unique and innovative solutions provide yet another, awesome set of play features to this and future iterations of <Adventure Game>.

DEVIATION FROM PROJECT PLAN

Team Rigel was very successful in achieving the goals set in the Project Proposal. We accomplished all of our Base and Extended Game goals. We had a few ideas reserved as stretch goals to pursue at a later date if 1) they fit the game’s personality and 2) if there was sufficient time prior to the deadline. We decided to implement the stretch goal of an independently moving enemy class with damage because we felt it would bring exceptional enrichment to the game experience, as well as an additional level of difficulty. Early in development, we entertained the idea of other stretch goals such as a timer which ran independent of the player’s input, a more complex system to color additional features on the map, or a save utility. However, in the final weeks of development, we decided that these stretch goals did not align with our vision for <Adventure Game> and so we decided not to pursue them.

CONCLUSION

The Capstone Project was a wonderful experience and a fantastic way to bring the Computer Science Post-Baccalaureate to a close. Not only did it provide valuable insight in planning and executing a large project within a set deadline, it also taught us all how to work both within and outside of the developing environment as a team. Each team member had the opportunity to both learn from and teach others, make and learn from mistakes in design and development, and find renewed confidence in our skills as programmers.

Looking back, we were so excited by the possibilities of what sort of fun and creative themes we could implement in our game, I think we really expected the foundation for the game as a whole to develop a bit more quickly and smoothly than it did. As the project grew in size, we also learned valuable lessons about taking extra care for ensuring your code is always scalable and easy to refactor without having to rewrite or write around what may have previously seemed like an inconspicuous and innocent element of hard coding early in development.

What is striking however is how much we learned about not only our weaknesses, but our strengths in this process. We learned patience in sludging through the “boring” parts of development to get to the fun bits. We leveled up our perseverance by repairing and reworking code when, in light of the unavoidable learning pains of version control when multiple people handle a project of this size, content was lost or altered. We exercised leadership and an exceptional team mentality when teaching, assisting, and reaching out for help help as we encountered challenges in development. And finally, we learned that we are incredibly competent in ways we did not know we were before Capstone. To our pride, when we dreamed up big, incredible rooms or features which we had no idea how to actually execute, rather than passing the idea off as too hard, we hit our development tools and figured it out. We came into Capstone eagerly looking ahead at our desire to become developers in the future. And now we leave Capstone, realizing that we already are. We hope you enjoy playing <Adventure Game> as much as we enjoyed creating it.

RESOURCES

ncurses

ncurses library

Creating an Adventure Game with ncurses

C++11

- [C++11]
- [Makefiles]
- [G++]