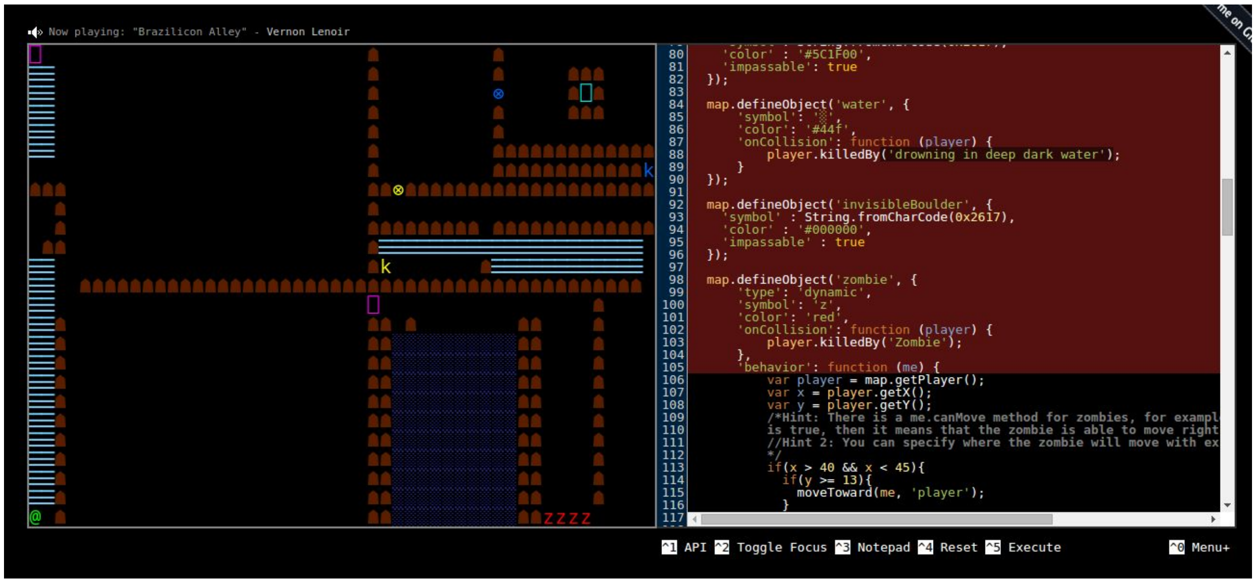
Alexander Lim, 34382135
Alexander Swainson, 38073128
Michael Adria, 39636113

CPEN 321 Team 15 - Lab Project Report

Untrusted - Labryinth Level


Now playing: "Brazilicon Alley" - Vernon Lenoir
^1 API ^2 Toggle Focus ^3 Notepad ^4 Reset ^5 Execute
^0 Menu+

```
    'color' : '#5C1F00',
    'impassable': true
});

map.defineObject('water', {
    'symbol': '░',
    'color': '#44f',
    'onCollision': function (player) {
        player.killedBy('drowning in deep dark water');
    }
});

map.defineObject('invisibleBoulder', {
    'symbol' : String.fromCharCode(0x2617),
    'color' : '#000000',
    'impassable' : true
});

map.defineObject('zombie', {
    'type': 'dynamic',
    'symbol': 'z',
    'color': 'red',
    'onCollision': function (player) {
        player.killedBy('Zombie');
    },
    'behavior': function (me) {
        var player = map.getPlayer();
        var x = player.getX();
        var y = player.getY();
        /*Hint: There is a me.canMove method for zombies, for exampl
        is true, then it means that the zombie is able to move right
        //Hint 2: You can specify where the zombie will move with ex
        */
        if(x > 40 && x < 45){
          if(y >= 13){
            moveToward(me, 'player');
          }
```

Introduction

For this project, we extended the open-source browser-based JavaScript adventure game called Untrusted. The programming language used is called “JSX” which is a wrapper language that compiles into JavaScript. It is faster, safer and easier to use. The runtime environment is Node.js and uses “http-server” for local development.

Link to the original project on GitHub: https://github.com/AlexNisnevich/untrusted.

Link to our forked project on GitHub: https://github.com/alexsclim/untrusted

Each level is initially unbeatable. The player can view the JavaScript code that creates the level, and can edit certain sections of it. By solving the code puzzle, the player can turn an impossible level into a beatable one. Our project was to add a bonus level with a high degree of difficulty, and involving many different elements of the original game. We named this level “Labryinth” based on the difficulty.

Our group agreed to to do a browser-based JavaScript project due to its wide use in web development. We were particularly interested in this game due to the use of JavaScript coding as a game mechanic. We were inspired by the more challenging levels of the game to make our level incorporate several different mechanics in order to make a level that required both careful understanding of the code and the game itself to complete.

By the end of the project, we had a fully finished, polished and working bonus level with all the different levels of complexity that we had intended to add to the level. This contributed to the system a new level which can be added as a bonus level to the systems existing ones.

Work Accomplished

To develop our level, we first forked the original project into our own repository, then set up the development environment using Node.js and npm. We had to learn JSX (JavaScript with additions), as well as the existing codebase. By the end of our project, we had successfully created a pull request containing our bonus level, ready to be merged into the original project.

Our level included a variety of gameplay mechanics:

- Icy ground, which prevented the player from controlling their movements.
- Gravity, which trapped the player and 'killed' them.
- Teleporters, which allowed the player to bypass the gravity section.
    - Initially, we wanted to use a 'telephone' item which would allow us to turn gravity on or off, but that proved to be a technical challenge.
- Zombies, which kill the player if touched.
- River crossings, which are passed by editing the code to place boulders to walk on.
- Locks and keys.

We incorporated many important software engineering practices discussed in the course throughout our project. We created requirements in order to make the level challenging, but beatable, and researched game mechanics to do so. We created scenarios based on user actions, and decided how the game should behave in each situation. With that, we sketched a rough design of our level, then began implementation. We constantly tested every component before moving on, allowing us to make sure we didn't add anything we would have to remove later. We used git branching to protect the master branch, and to allow other members to do 'code reviews', although our practice of pair programming did reduce the need for this. At some points, we had to change the project requirements as some features did not fit the scope of our whole project.

Our development process encompassed the ideas of agile development as it was more flexible and requirements changed many times. Prior to submitting the pull request, we had test cases to verify the number of objects being created on our map during runtime. On each area of the map in which the user could parish (gravity, water, zombies), our code would verify that the user would die when the correct conditions were met. After we had finished implementing the bonus level, we refactored our code to keep a clean and consistent styling with respect to the original code base.

As part of our acceptance tests, each of us had to solve an entire runthrough of the map by ourselves. We had worked in pairs on different parts of the map, but no one of us knew the solution to the entirety of the map. If one of us found that there were difficult parts which took a while to solve, we would change our code to make the map easier, or include hints as part of comments in the editable sections of the code. A notable part where our acceptance tests failed was with the moveable zombies. Without the hint which we later incorporated into the editable code section, that portion of the map would've been nigh impossible to solve.

In our project, we had added 308 lines of code while modifying 1 line of code to include our bonus level in the project. There were some test cases to make sure the proper amounts of each object had shown up on the map. This proves to be sufficient as the bonus level is primarily composed of adding new objects to the map and designing them in a way so that it is difficult for the user to solve. The original system has no unit testing framework and due to the nature of the browser based game, there was no way to show test coverage of our test cases as the best way to test the game was to actually play the game itself.

Each group member took part in every step in the development process. Requirements were made during a group meeting and scenarios and designs were also done during that time. For development, we decided it would be most effective to pair program and just test the game together as we go. This allowed every member of the group to have their input during the development of the game. Each member had an input on the map as we built it and contributed ideas during our pair programming sessions until the bonus level was done.

Conclusion

Contributing to this project provided valuable experience with different Software engineering practices. Our group decided on an Agile methods, to best fit the constantly changing requirements we wanted for our level, as we found different, interesting gameplay experiences across the provided levels. Pair programming was another major development style we choose to follow, because it would increase the ease of performing code reviews, easily allow for us to have a consistent code design across the entire level. While one person would write the code to populate the map elements and their behaviours. the other person could theorize which portions of the code could altered to solve that portion of the map. Prior to submitting the pull request, we had several tests to verify expected behaviours, however we removed them prior to submitting our pull request, to keep with the style of the repository.

When we were finished with our level, it had included many of the other difficult aspects of the prior maps in game, as well as creating some new ones: sliding ice, gravity, moving enemies, teleportation, and several more. By the end of the project, we had contributed to the system a new level which can be added as a bonus level to the system's existing ones. Our completed bonus level is now ready for merging into the original project if approved by the owner of the Untrusted game.

Deliverables

Deliverables for the project include: This project report, our GitHub repository containing all our code, and a demonstration of our bonus level.