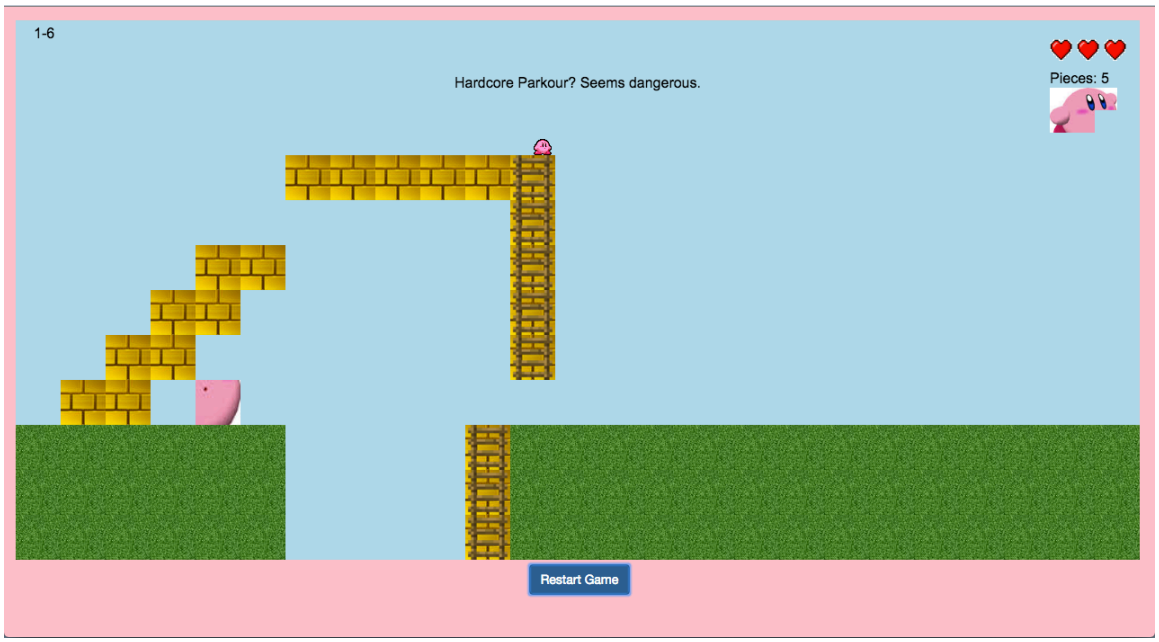
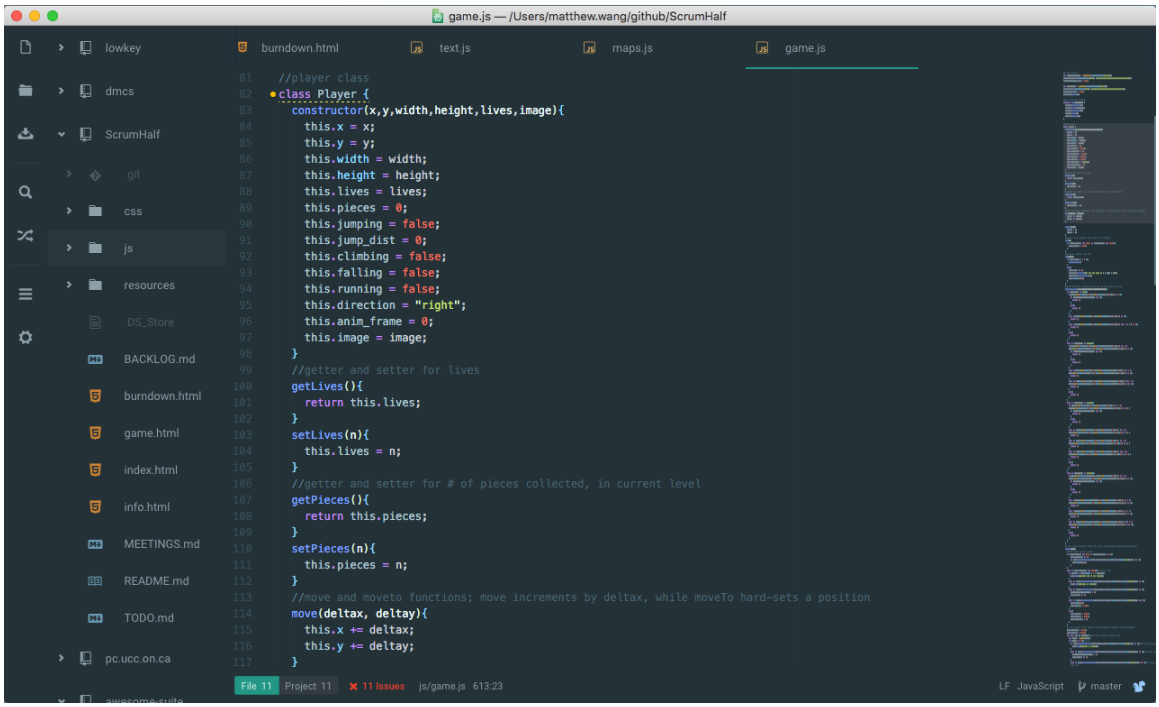
Scrum Half

A game by Matthew Wang, Isaac Ng, Saeed Khan, and Aidan Heney

A retrospective report by **Matthew Wang**



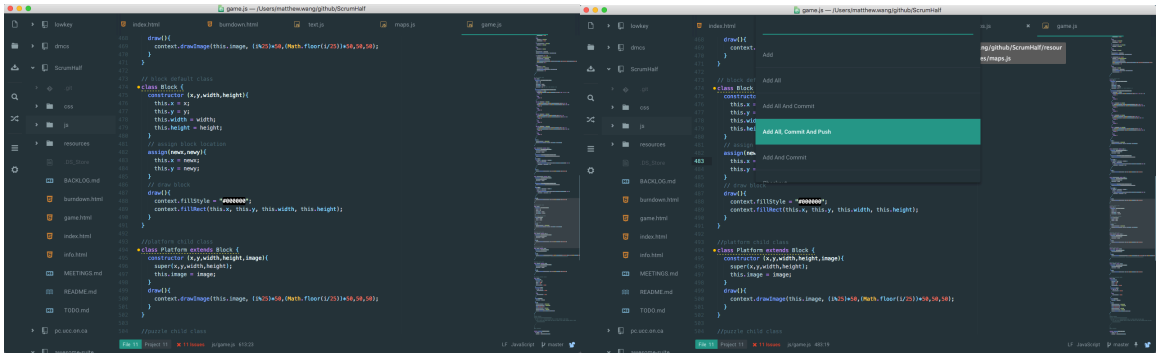


The Workspace

Desktop

Atom

Atom is my text-editor of choice: it includes a lot of modularity, like syntax highlighting, linting, GitHub integration, autocomplete, and bracket-matching; and also looks pretty cool, and runs on every OS. Here's a picture of what my workspace normally looks like.

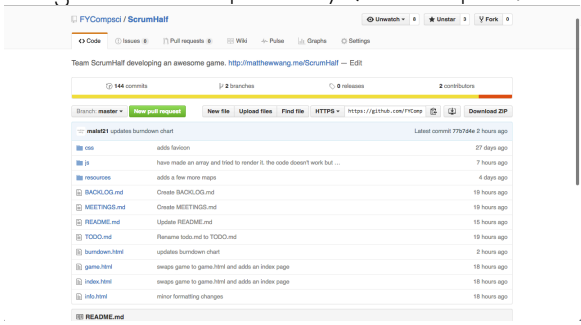


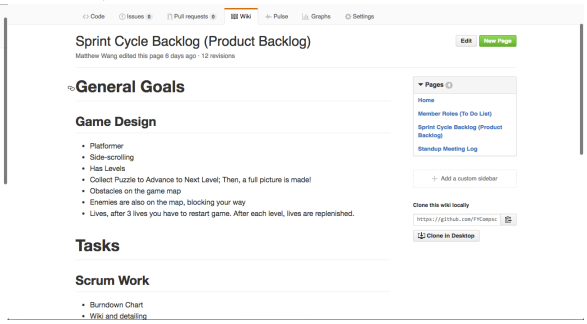
I also run all of my Git commands through here, like pushing, pulling, and merging branches.

Online

GitHub.com

GitHub is our code version control software (CVC for short). I've hooked Atom into it, but it also has an online version. Here, you can browse through all files on the repo, as well as make edits, make pull requests, issues, and wiki pages. Here's a few images of our repository (FYCompSci/ScrumHalf):

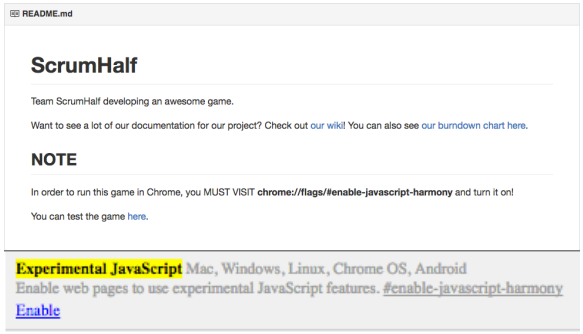




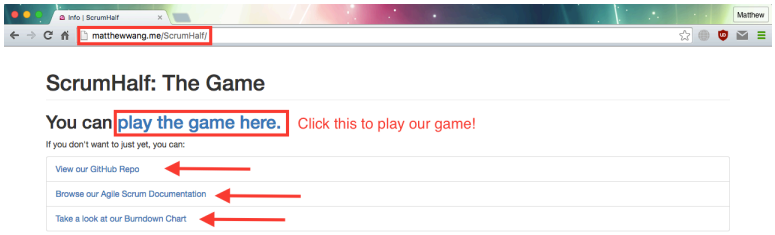
I'll talk more about what we did with GitHub in the Development Process section.

The Game
Showcase
Starting the Game

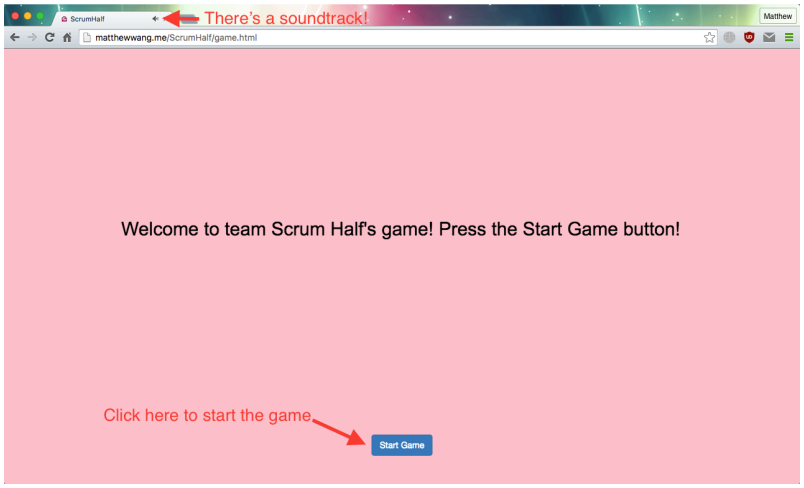
Since our game uses ECMA6, you'll need to enable it in your browser. The latest build of Firefox should have it, while Chrome has an option to enable it.



Then, you'll want to visit our website, which is online. It's at http://matthewwang.me/ScrumHalf.



You can either play our game, look at our repo, look at our documentation, or take a look at our burndown chart. Let's start the game for now.

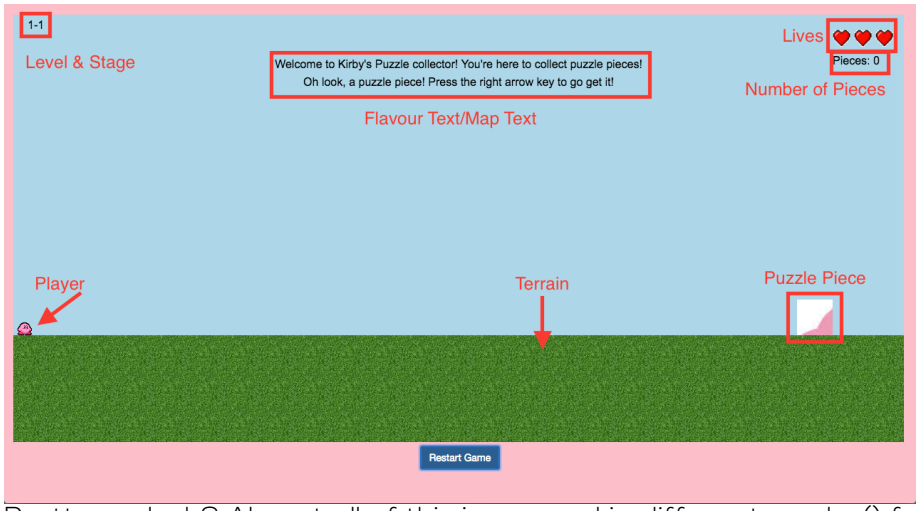


Now, you're all set! Let's jump into this game. Pressing start will also change the music, from the loading music to some wonderful in-game tunes.

Playing the Game and Features

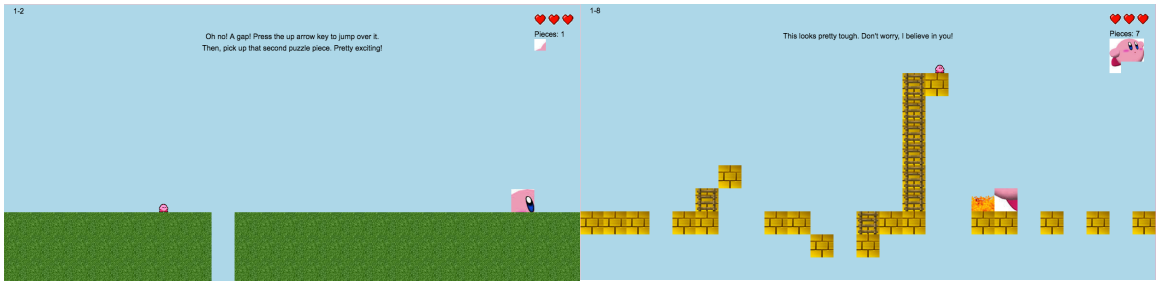
In order to move your character around, you can use the arrow keys to run and jump. The character's portrait will change based on its actions. Examples include  or  . Since we use a spritesheet system, there is an infinite possibility of amount of frames, and objects available.

Let's take a look at the Game's UI.



Pretty cool, eh? Almost all of this is covered in different render() functions.

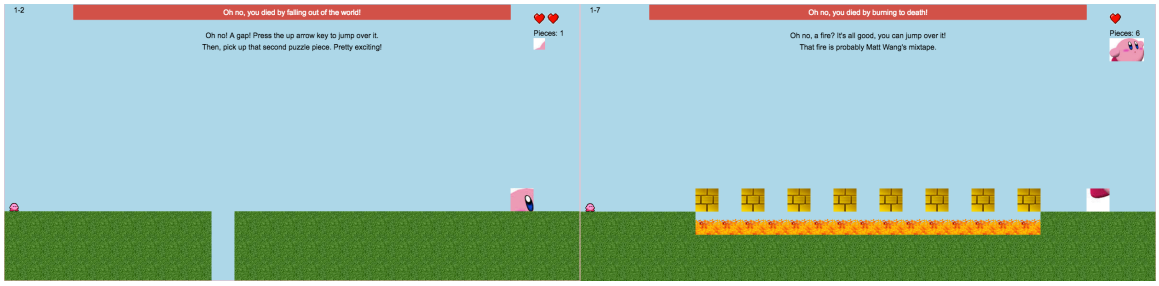
A quick note: the puzzle pieces collected will create a portrait, like so. Wonder who that might be!



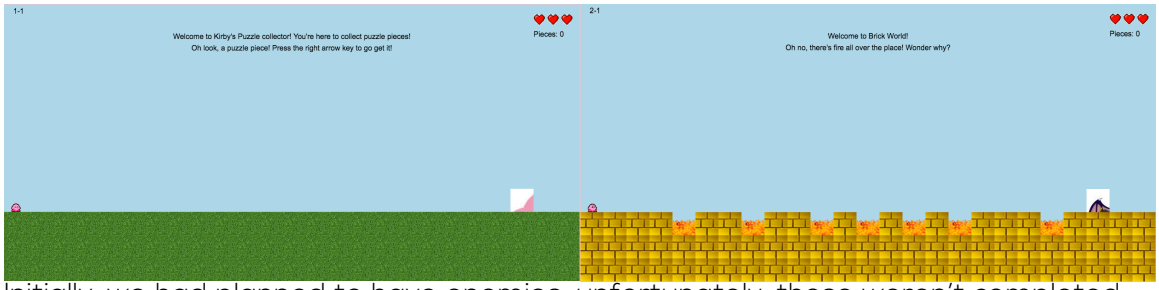
There are several terrain objects, including grass, bricks, fire, ladders, and puzzle pieces. Ladders, fire, and puzzle pieces are interactive: you can climb ladders, die to fire, and collect puzzle pieces.

The objective of the game is to collect all the puzzle pieces in one level (there are 9 puzzle pieces, as well as 9 stages per level). Once all the pieces have been collected in one level, you can move on to the next one.

Players can die by falling out of the world or hitting fire. Doing so pops an alert, explaining your death.



There are two levels: Grass World, and Brick World.

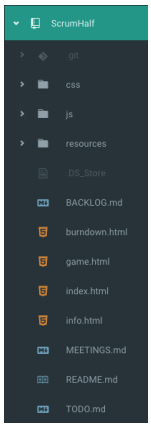


Initially, we had planned to have enemies; unfortunately, those weren't completed.

That's the game! Obviously, playing it is the best way to experience it: you can play it at http://matthewwang.me/ScrumHalf

Code

Overview

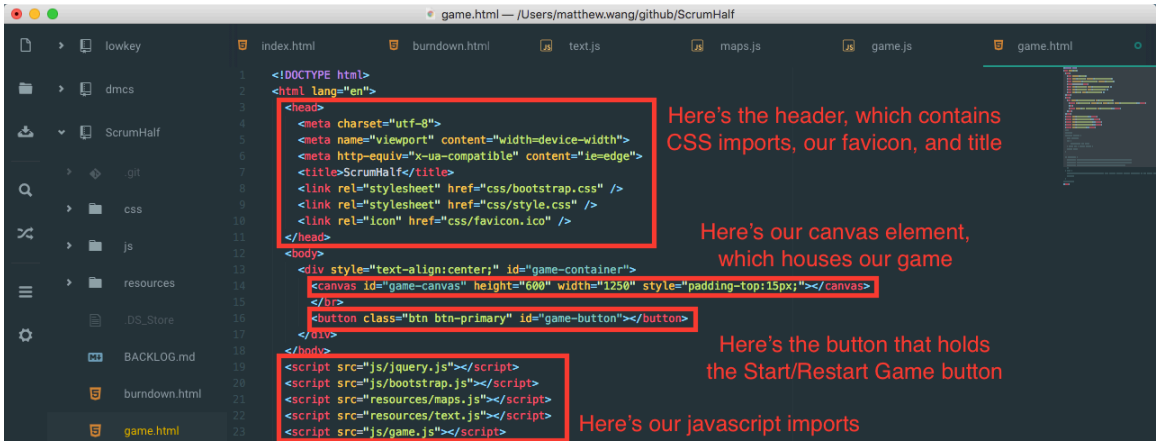


Here's the filepath of our project. We have a few .html pages, including the homepage, the game container, an information page, and our burndown chart; a css folder, which contains bootstrap and some custom css; the js folder, that contains our game's javascript, as well as bootstrap, jQuery, marked, and our graphing library; the resources folder, which contains all of our images, music, and spritesheets; and a few Markdown files that are displayed on the website.

Our game is located at game.html, while the other pages include other info.

HTML

Let's take a quick look at what we used to make the game. First top, game.html.



Nothing too complicated, mostly just standard HTML structure, a canvas element, and a button. Interestingly, the button doesn't have any contents or an onclick event; instead, I've used jQuery to add those attributes depending on the game state.

CSS

While we didn't really end up using too much CSS, since our game was almost entirely in the canvas element, we did use it a few times to format things properly. Let's take a look!

```
<style>
  .scrum{
    height:256px;
    overflow-y: scroll;
  }
</style>
```

```
body{
  background-color: pink;
}
```

The first section of code is the styling for a scrollbox (which is located at info.html). It sets a fixed height, then forces a scrollbar for any other overflows. The second snipped of CSS sets the background colour to pink, which is what we used in for the game's background page, as well as our burndown chart's background page.

Javascript: Arrays and Mapping

During the planning stage of this game, we figured out that we need to have some sort of system for a map. I was the main coder of that area, and I opted to use multi-dimensional arrays. Let's take a look!

```
var world1_map = [
  [//level 1
    [//stage 1
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,
      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
    ],
    [//stage 2
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,0,0,
      1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
      1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
      1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
    ],
```

```
var world1_text = [
  [//level 1
    [//stage 1
      ["Welcome to Kirby\'s Puzzle collector! You're here to collect puzzle pieces!", 625, 75],
      ["Oh look, a puzzle piece! Press the right arrow key to go get it!", 625, 100]
    ],
    [//stage 2
      ["Oh no! A gap! Press the up arrow key to jump over it.", 625, 75],
      ["Then, pick up that second puzzle piece. Pretty exciting!", 625, 100]
    ],
    [//stage 3
      ["Some platforms? Wonder if you can jump over them?", 625, 75]
    ],
    [//stage 4
      ["Oh cool, a ladder! Just don't walk under it!", 625, 75]
    ],
    [//stage 5
      ["Oh snap, that platform might need some repairs. Jump in the meantime.", 625, 75]
    ],
    [//stage 6
      ["Hardcore Parkour? Seems dangerous.", 625, 75]
    ],
    [//stage 7
      ["Oh no, a fire? It's all good, you can jump over it!", 625, 75],
      ["That fire is probably Matt Wang's mixtape.", 625, 100]
    ],
    [//stage 8
      ["This looks pretty tough. Don't worry, I believe in you!", 625, 75]
    ],
    [//stage 9
      ["Wow, seems like the last puzzle piece. I wonder what happens if you collect it?", 625, 75]
    ]
  ],
```

Here's a small snippet of the world1 map, and the text for those levels. It looks somewhat similar to the actual world map! While there's some enter formatting, it's really just one long list of 0's, 1's, and any other integer. Each number represents a different type of block: 0 represents nothing, 1 a grass block, 4 a puzzle block, etc. The render map function then takes that, and draws the corresponding item.
But how does the computer now when there's a new row or column? Well, in order to figure out what "row" a specific item is in, you can use the modulus operator (%). If you're trying to figure out what 'column" a specific item is in, you simply use Math.floor, which is very similar to Python or Java's divide functions. That's used later throughout this report.

In addition, we used this same principle to render text on the screen. Both of these approaches combined allow users to create their own worlds, and allows for a lot more modularity and ease-of-use in the code design

Javascript: Animating Frames

The first step to any game like this is animating frames on the page. Let's take a look at the segments of code where we render the frames.

```
//initial animate function
var animate = window.requestAnimationFrame || window.webkitRequestAnimationFrame || window.mozRequestAnimationFrame ||
function (callback) {
  window.setTimeout(callback, 1000 / 60);
};
```

This first piece of code defines animate, which is a function that occurs 60 times a second. It acts as a pseudo-framerate.

```
var update = function(){
  updateButton();
  if (started === true){
    updateMap(1,level,stage);
    player.update(levelMap);
  }
};

var render = function () {
  if (started === false){ //start game screen
    context.textAlign = "center";
    context.fillStyle = "pink";
    context.fillRect(0, 0, cwidth, cheight);
    context.fillStyle = "#000000";
    context.font = "30px Arial";
    context.fillText("Welcome to team Scrum Half's game! Press the Start Game button!",625,285);
  }
  else{

    context.fillStyle = "lightblue";
    context.strokeStyle = "lightblue";
    rect(0,0,cwidth,cheight);

    //context.drawImage(skyBackgroundImage,0,0,cwidth,cheight);
    renderMap(levelMap);
    player.draw();
    renderText();
    renderLives();
    renderPuzzle(puzzleMap,puzzleImage);
  }
};

var step = function () {
  update();
  render();
  animate(step);
};
```

Here, we have our logic loop that puts things on the screen. First, we update the states of things that need to be updated (in this case, that would be the map and the player). After that, we then render everything we need to: this includes drawing the sky, each type of block, the UI, and the player.

```
function renderText(){
  context.fillStyle = "black";
  context.font = "16px Arial";
  context.textAlign = "start";

  context.fillText(String(level+1) + "-" + String(stage+1),20,20);
  context.fillText("Pieces: " + player.getPieces(),1150,70);

  context.textAlign = "center";

  for (i = 0; i < world1_text[level][stage].length; i ++){
    context.fillText(world1_text[level][stage][i][0],world1_text[level][stage][i][1],world1_text[level][stage][i][2]);
  }

  if (alertText.activated === true){
    alertText.draw();
  }
}

//other UI functions

function renderLives(){
  for (i = 0; i <3; i ++){
    if (player.getLives() > i){
      context.drawImage(lifeHeartImage, 1150+i*30,20);
    }
  }
}

function renderPuzzle(puzzleMap,puzzleImage){
  for (i=0;i<9;i++){
    if (puzzleMap[level][i] == 1){
      context.drawImage(puzzleImage,i%3*50,Math.floor(i/3)*50,50,50,1150+i%3*25,75+Math.floor(i/3)*25,25,25);
    }
  }
}
```

Here's a few examples of the available render functions. All of them check for conditions, and draw different things to the canvas based on whether or not those conditions are fulfilled. Note the use of canvas methods like drawImage().

```
function renderMap(map){
  for (i = 0; i < map.length; i++){
    if (map[i] == 1){
      grassBlock.assign((i%25)*50,(Math.floor(i/25))*50);
      grassBlock.draw();
    }
    else if (map[i] == 2){
      goldBlock.assign((i%25)*50,(Math.floor(i/25))*50);
      goldBlock.draw();
    }
    else if (map[i] == 3){
      goldBlock.assign((i%25)*50,(Math.floor(i/25))*50);
      goldBlock.draw();
      ladderBlock.assign((i%25)*50,(Math.floor(i/25))*50);
      ladderBlock.draw();
    }
    else if (map[i] == 4 && puzzleMap[level][stage] === 0){
      puzzleBlock.assign((i%25)*50,(Math.floor(i/25))*50);
      puzzleBlock.draw();
    }
    else if (map[i] == 5){
      fireBlock.assign((i%25)*50,(Math.floor(i/25))*50);
      fireBlock.draw();
    }
    /*
    else{
      context.fillStyle = "lightblue";
      context.strokeStyle = "lightblue";
      rect((i%25)*50,(Math.floor(i/25)*50),50,50);
    }
    */
  }
}
```

Here, we end up checking each element in the map array for it's specific value, and then run the corresponding draw area for the specific array element, using the modulus and Math.floor operators.

Javascript: Classes and ECMAScript6

In ECMAScript5, the base for the most popular/stable version of Javascript right now, classes look very different from what we're familiar with in Java. Here's a comparison of what ECMA5 vs. ECMA6 looks like; both are my projects. (Left is ECMA 5, right is ECMA6).

```
function Paddle(x, y, width, height) {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.x_speed = 0;
    this.y_speed = 0;
}

Paddle.prototype.render = function () {
    context.fillStyle = "#FFFFFF";
    context.fillRect(this.x, this.y, this.width, this.height);
};

Paddle.prototype.move = function (x, y) {
    this.x += x;
    this.y += y;
    this.x_speed = x;
    this.y_speed = y;
    if (this.x < 0) {
        this.x = 0;
        this.x_speed = 0;
    } else if (this.x + this.width > 400) {
        this.x = 400 - this.width;
        this.x_speed = 0;
    }
};

Paddle.prototype.resetX = function (){
  this.x = 175;
  this.x_speed = 0;
};
```

```
// block default class
class Block {
  constructor (x,y,width,height){
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
  }
  // assign block location
  assign(newx,newy){
    this.x = newx;
    this.y = newy;
  }
  // draw block
  draw(){
    context.fillStyle = "#000000";
    context.fillRect(this.x, this.y, this.width, this.height);
  }
}

//platform child class
class Platform extends Block {
  constructor (x,y,width,height,image){
    super(x,y,width,height);
    this.image = image;
  }
  draw(){
    context.drawImage(this.image, (i%25)*50,(Math.floor(i/25))*50,50,50);
  }
}
```

Classes are a pretty important part of Computer Science and Object-Oriented Programming, and they're super useful! Since in this group project our other members aren't familiar with ECMA5's prototype method, we switched to ECMA6 and have some nice classes. For example, the Block class above acts as a base parent class, and from that we get other blocks such as the mentioned Platform child-class, Fire child-class, etc. Classes are a big part of our project: OOP is definitely one of the core principles we use! In addition to Blocks, we also use a hierarchy of classes for Players, and Alerts, and have custom methods from them.

In addition, we adhere to some OOP standards. For example, we use setters and getters for class-related variables, as shown below.

```
class Player {
  constructor(x,y,width,height,lives,image){
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.lives = lives;
    this.pieces = 0;
    this.jumping = false;
    this.jump_dist = 0;
    this.climbing = false;
    this.falling = false;
    this.running = false;
    this.direction = "right";
    this.anim_frame = 0;
    this.image = image;
  }
  //getter and setter for lives
  getLives(){
    return this.lives;
  }
  setLives(n){
    this.lives = n;
  }
  //getter and setter for # of pieces collected, in current level
  getPieces(){
    return this.pieces;
  }
  setPieces(n){
    this.pieces = n;
  }
  //move and moveto functions; move increments by deltax, while moveTo hard-sets a position
  move(deltax, deltay){
    this.x += deltax;
    this.y += deltay;
  }
```

Our player class itself is huge, hitting just over 258 lines of code! That being said, it makes it easier for us to program future goals if needed.

Javascript: Spritesheets!

```
draw(){
  var srcString = "resources/player/kirby";
  var needs_sprite = false;
  if (this.jumping === true || this.climbing === true){
    srcString += "Up";
  }
  else if (this.falling === true){
    srcString += "Down";
  }
  else if (this.running === true){
    srcString += "Run";
    needs_sprite = true;
  }
  else{
    srcString += "Normal";
    needs_sprite = true;
  }
  if (this.direction == "right"){
    srcString += "Right";
  }
  else{
    srcString += "Left";
  }
  this.image.src = srcString;
  context.drawImage(this.image, this.x,this.y);

  if (needs_sprite === true){
    srcString += "Spritesheet.png";
    this.image.src = srcString;
    context.drawImage(this.image,
    (Math.floor((this.anim_frame)/30)%(this.image.width/32))*32,0,32,32,this.x,this.y,32,32);
    this.anim_frame += 1;
  }
  else{
    srcString += ".png"
    this.image.src = srcString;
    context.drawImage(this.image, this.x,this.y);
```

Getting just a static image is kind of boring, to be quite honest. Having a character blink is super cool, and that's exactly something we wanted to do! In the Player.draw() function, that's shown above, we go through a way to parse spritesheets like the one shown below into running, breathing, and jumping animations. Pretty cool stuff!

General Notes and Observations

While I'm already experienced with Javascript and web-design, there's a few nice skills and thoughts that I've picked up throughout this coding jam. This includes:

- Using Math.floor(); in Javascript, the "/" operand returns a decimal, not a whole number like in languages like Python and Java.
- For and If logic statements are very similar to their Java counterparts, as they are both heavy C-based languages.
- Classes in this project were very useful, especially instantiating many instances of classes at the same time, and using class-specific methods.
- Gravity is hard! One of the largest challenges with this project was player falling and jumping: it took almost 2 weeks to sort out just that problem! Luckily, with some collaboration and testing, we figured it out.
- Well-organised code is super important! Collaborating with others emphasized how important good documentation is.

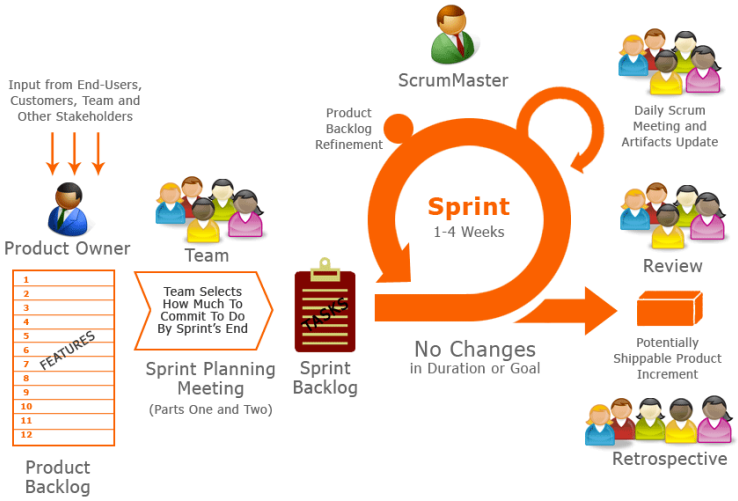
There's a lot more we've done in this game, including updating logic statements, using jQuery, audio, drawing state-based images, and map switching, but unfortunately there's a limited amount of time and space. That being said, you can visit our github repository (which you can find on http://matthewwang.me/ScrumHalf/) to see more of our code.

The Scrum

Overview

Overview

Agile Scrum is an SDLC (Software Development Life Cycle) system that prioritizes short cycles of working on specific issues versus large, segmented planning stages that are often common in the Waterfall SDLC. In addition, it focuses on getting out a testable, working product as soon as possible, versus waiting for a long time for a finished product. In our project, we used Agile Scrum: partially to learn how it works, but also because it seems pretty cool! Here's a description of how we ended up using Agile Scrum to our advantage.



Agile Scrum contains artefacts, or items used to aid the development. One of them is the Product Backlog is a prioritized features list: containing everything that a project should complete in its entirety. This is set at the beginning of the entire operation, though it may be edited.

In Agile Scrum, time periods are split up into two or three week "Sprints", where set objectives are completed in each Sprint. This list of things to-do in the Sprint is often called a Sprint Backlog. At the end of a Sprint, there's a Sprint Review meeting: there, the Scrum Master (the development team's scrum project manager) reviews what has or has not been completed, and often demos the completed work.

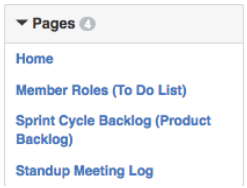
Each day the team meets for work is called a Scrum. A Scrum first begins with a standup meeting: the Scrum Master, a project manager for the development team, goes over what has been done and what needs to be done in the current Scrum. Then, the team codes, and tests their code.

A Burndown Chart is another artefact of Agile Scrum: it documents the amount of time left versus the amount of tasks left to complete, either for a Sprint or for the entire project. Often, it will also include a prediction for how much work should be completed by the end of said time period.

Planning

Product (and in this case, Sprint) Backlog

In our situation, the Sprint/Product Backlogs were the same thing (since we only had one Sprint). At the inception of our organisation, we created a Product Backlog of 20 items. They're displayed below, but can also be accessed at http://matthewwang.me/ScrumHalf/info.html



The menu for our wiki.

Product/Sprint Backlog

Scrum Work

- Burndown Chart
- Wiki and detailing

Coding

- Object-Oriented Design & ECMA6 Classes
- Start Game/Reset Button
- Player Movement (Left, Right , Jump/Up, Collision Detection). Requires player class.
- Platforms (placed collision detection obstacles). Requires platform class.
- Ladders (Moving up and down blocks). Requires Block class.

- Obstacles (Obstacles that stop player movement). Requires Block Class.
- Enemies (spawned moving obstacles). Requires enemy base class, and possible subclasses.
- Puzzle pieces (spawned collectible objects). Requires puzzle piece class, and possible subclasses.
- Map Rendering (Takes in array and renders map)
- Text Rendering (Takes in array and renders text)
- Map-changing (Edge of the map equals new loaded map)
- Lives and dying

Resources

- Player Design
- Puzzle Pieces
- Non interactive Map Elements (Platforms, Blocks, etc.)
- Interactive Map Elements (Enemies)
- Music
- Resource Bar (Lives, Puzzle Pieces)

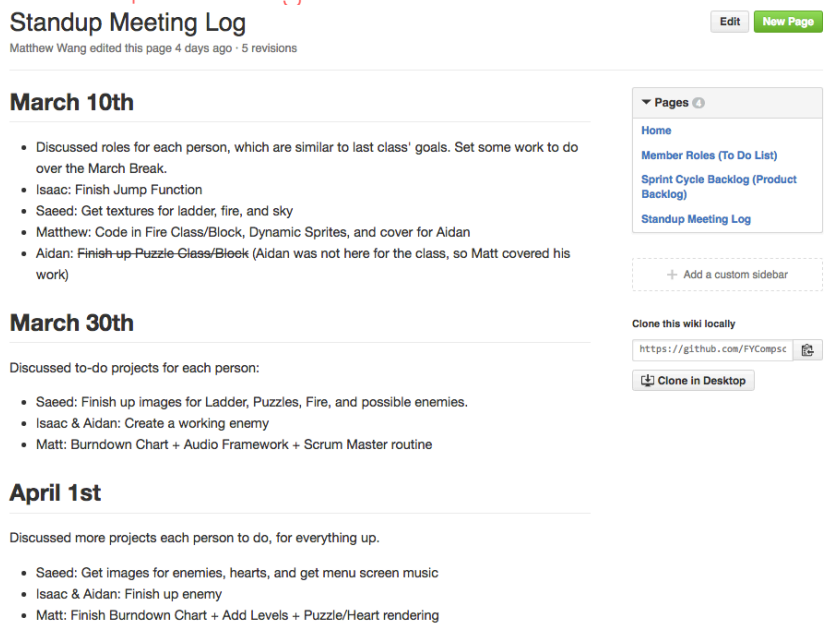
Member Roles

In every team, each person has a role. In our situation, we set some designated roles for each member. They are:

- Matthew Wang: Scrum Master (managing tasks, product backlog, burndown charts, team management), Assistant Developer (helps out with game development, and creates webpages)
- Isaac Ng: Lead Player Developer (develops code for player interactions)
- Aidan Heney: Lead Environment Developer (develops environment interactions)
- Saeed Khan: Resource Manager (Finds images, music, and spritesheets)

Sprinting!

Standup Meetings



Standup Meeting Log

Edit New Page

Matthew Wang edited this page 4 days ago · 5 revisions

March 10th

- Discussed roles for each person, which are similar to last class' goals. Set some work to do over the March Break.
- Isaac: Finish Jump Function
- Saeed: Get textures for ladder, fire, and sky
- Matthew: Code in Fire Class/Block, Dynamic Sprites, and cover for Aidan
- Aidan: ~~Finish up Puzzle Class/Block~~ (Aidan was not here for the class, so Matt covered his work)

March 30th

Discussed to-do projects for each person:

- Saeed: Finish up images for Ladder, Puzzles, Fire, and possible enemies.
- Isaac & Aidan: Create a working enemy
- Matt: Burndown Chart + Audio Framework + Scrum Master routine

April 1st

Discussed more projects each person to do, for everything up.

- Saeed: Get images for enemies, hearts, and get menu screen music
- Isaac & Aidan: Finish up enemy
- Matt: Finish Burndown Chart + Add Levels + Puzzle/Heart rendering

Pages

Home

Member Roles (To Do List)

Sprint Cycle Backlog (Product Backlog)

Standup Meeting Log

Add a custom sidebar

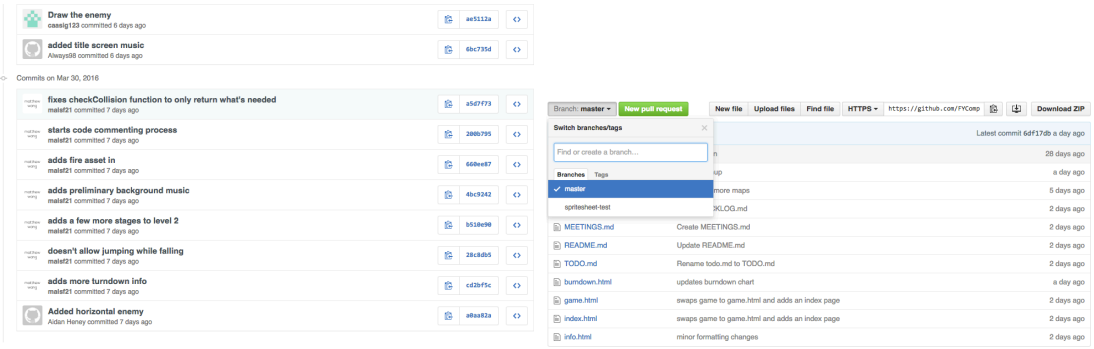
Clone this wiki locally

https://github.com/FYCompsc

Clone in Desktop

As the Scrum Master, I lead standup meetings at the beginning of every Scrum (and also took note of what we discussed). We set goals for the next class/Scrum, and record them in the Standup Meeting Log as well as the To Do List in the Member Roles. If you'd like to browse our wiki and see more of our process, you can at http://github.com/FYCompSci/ScrumHalf/wiki

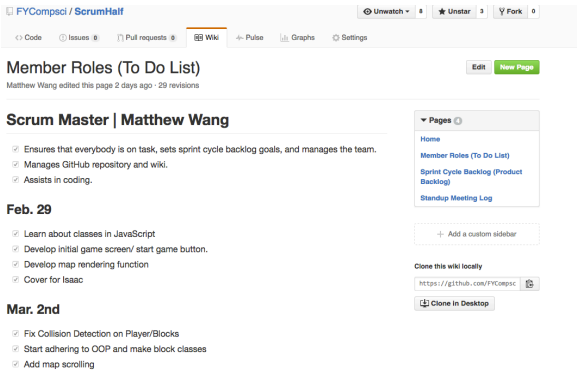
Using GitHub

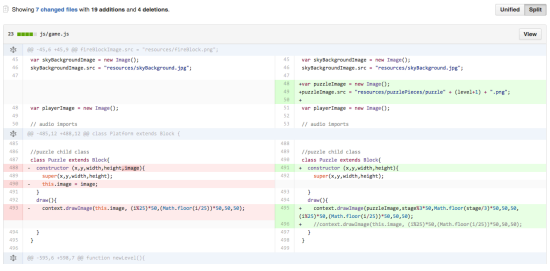


During our Scrums, our developers used GitHub effectively. There are a few areas that using GitHub helped out our developing process (some of which is shown above)

- GitHub made it easy for us to organize and keep our code updated to the latest version
- GitHub's merging system allowed our developers to work on two different parts of the code at the same time, without having merge conflicts
- GitHub's branch system allowed us to try out new features without impacting others' code in the progress, and also allowed us to test out new features quickly but safely
- Combining GitHub and our GitHub wiki made it easier for contributors to understand what needed to be done.
- GitHub's commit history system helped us bugfix, by figuring out when a specific line of code was changed

GitHub helped increase our productivity throughout the entire Sprint! Here are some more examples of how we used GitHub.





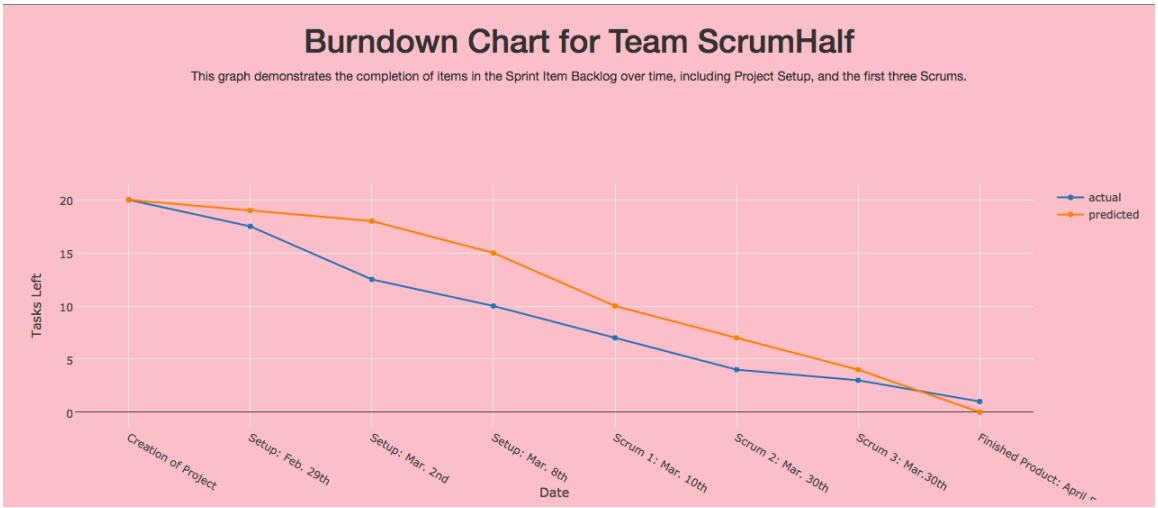
Sprint Review

Sprint Review Meeting

After we'd finished as much as we could, we held a Sprint Review meeting, We looked over what we'd completed (which was an awesome amount), what we ended up not completing (in this case, just the Enemy concept), and did a little demo run of our project. As well, we discussed what our team did well (and what we unfortunately didn't), linking with the start-stop-continue methodology. Things we discussed included hitting deadlines, creating easily readable code, learning how to bugtest, and using GitHub effectively (branches, commits, and the wiki).

Burndown Chart

The Burndown Chart displays the completion of tasks during the sprint. Unfortunately, we didn't reach our end goal, but we were quite close. You can find the Burndown Chart online at http://matthewwang.me/ScrumHalf/burndown.html



As you can see, our actual tasks completed beat our predictions until the very end. In the Sprint Review Meeting, we were pretty happy about the work we've done! That last task that we hadn't completed was the Enemies, but those aren't core to our game or its design.

We used plot.ly JS, a JavaScript graphing library, to make this interactive graph.

```
var xValue = ["Creation of Project", "Setup: Feb. 29th", "Setup: Mar. 2nd", "Setup: Mar. 8th", "Scrum 1: Mar.
10th", "Scrum 2: Mar. 30th", "Scrum 3: Mar.30th", "Finished Product: April 5th"];

var yValue1 = [20,17.5,12.5,10,7,4,3,1];
var yValue2 = [20,19,18,15,10,7,4,0];

var trace1 = {
  x: xValue,
  y: yValue1,
  type: 'scatter',
  name: 'actual'
};

var trace2 = {
  x: xValue,
  y: yValue2,
  type: 'scatter',
  name: 'predicted'
};

var data = [trace1,trace2];

var annotationContent = [];

var layout = {
  xaxis: {
    title: 'Date'
  },
  yaxis: {
    title: 'Tasks Left',
    showticklabels: 'false'
  },
  paper_bgcolor: 'rgba(0,0,0,0)',
  plot_bgcolor: 'rgba(0,0,0,0)'
};
```

The Reflection
Personal Reflection
Overview

I was pretty excited about this project: I have a lot of experience coding in projects and using GitHub, but the Agile Scrum methodology was new to me, and being Scrum Master was a big responsibility. At the end of the project, I'm very happy with what I've done: I refined a lot of my coding skills, and also learned a lot about Agile Scrum and how SDLCs work!

```
// creating on-screen alerts
class Alert{
  constructor(){
    this.text = "";
    this.x = 625;
    this.y = 25;
    this.activated = false;
  }
  setActivated(key){
    if (key === 0){
      this.activated = false;
    }
    else{
      this.activated = true;
    }
  }
  setText(newText){
    this.text = newText;
  }
  draw(){
    context.fillStyle = "#d9534f";
    context.strokeStyle = "#d9534f";
    rect(150,3,950,32);
    context.fillStyle = "white";
    context.strokeStyle = "white";
    context.fillText(this.text,this.x,this.y);
  }
}
```

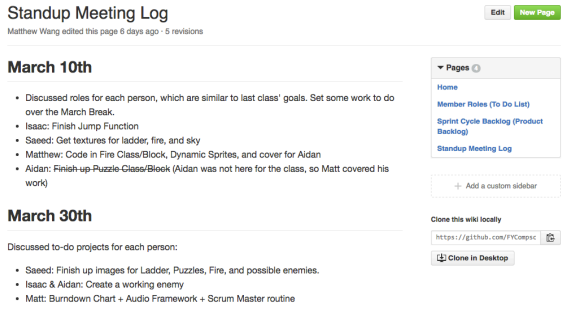
```
draw(){
  var srcString = "resources/player/kirby";
  var needs_sprite = false;
  if (this.jumping === true || this.climbing === true){
    srcString += "Up";
  }
  else if (this.falling === true){
    srcString += "Down";
  }
  else if (this.running === true){
    srcString += "Run";
    needs_sprite = true;
  }
  else{
    srcString += "Normal";
    needs_sprite = true;
  }
  if (this.direction == "right"){
    srcString += "Right";
  }
  else{
    srcString += "Left";
  }
  this.image.src = srcString;
  context.drawImage(this.image, this.x,this.y);

  if (needs_sprite === true){
    srcString += "Spritesheet.png";
    this.image.src = srcString;
    context.drawImage(this.image,
    (Math.floor((this.anim_frame)/30)%(this.image.width/32))*32,0,32,32,this.x,this.y,32,32);
    this.anim_frame += 1;
  }
```

Some examples of what I learned coding-wise: creating ECMA6 classes in Javascript, and working with spritesheets.

Contributions

During the planning phase, I was assigned the position of Scrum Master. Throughout the Sprint, I did my best as what a Scrum Master should do: holding standup meetings, documenting our progress, keeping members on task, and making the burndown chart. In addition, I would assign what work would need to be done each Scrum, and who needed to do it.




```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <meta http-equiv="x-ua-compatible" content="ie=edge">
    <title>Burndown Chart | ScrumHalf</title>
    <link rel="stylesheet" href="css/bootstrap.css" />
    <link rel="stylesheet" href="css/style.css" />
    <link rel="icon" href="css/favicon.ico" />
  </head>
  <body style="text-align:center;">
    <h1>Burndown Chart for Team ScrumHalf</h1>
    <p>This graph demonstrates the completion of items in the Sprint Item Backlog over time, including Project Setup, and the first three Scrums.</p>
    <div id="burndown-chart"></div>
  </body>

  <script src="js/jquery.js"></script>
  <script src="js/bootstrap.js"></script>
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
  <script>
    var xValue = ["Creation of Project", "Setup: Feb. 29th", "Setup: Mar. 2nd", "Setup: Mar. 8th", "Scrum 1: Mar. 10th", "Scrum 2: Mar. 30th", "Scrum 3: Mar.30th", "Finished Product: April 5th"];

    var yValue1 = [20,17.5,12.5,10,7,4,3,1];
    var yValue2 = [20,19,18,15,10,7,4,0];

    var trace1 = {
      x: xValue,
      y: yValue1,
      type: 'scatter',
      name: 'actual'
    };
```

Some examples of the work I did as the Scrum Master.

Ultimately, I think I ended up doing what I needed to do as Scrum Master, and drive the vision of the game in the product backlog forward, managing our team and resources.

In terms of coding, I was initially supposed to be the backup coder, in case something went wrong, or our developers were having trouble. Since the scope of our project was huge, and our other developers didn't know too much about Javascript, I helped write most of the code for the project, dealing especially with logic loops and the math required.

```
//check collision function, checks for up, down, left, and right
checkCollision(x,y,width,height,direction,map){
  if (direction == "up"){
    if(map[(Math.floor(x/50))+(Math.floor((y-1)/50))*25] == 4){
      if (puzzleMap[level][stage] === 0){
        return 4;
      }
      else{
        return 0;
      }
    }
    else if(map[(Math.floor(x/50))+(Math.floor((y-1)/50))*25] == 5){
      return 5;
    }
    else if(map[(Math.floor(x/50))+(Math.floor((y-1)/50))*25] === 0 && y -5 > 0){
      return 0;
    }
    else{
      return 1;
    }
  }
```

```
// this happens to set default states, instead of checking for key released
this.climbing = false;
this.running = false;
for (var key in keysDown) { // n-key rollover movement code
  var value = Number(key);
  if (value == 38){ // up
    if (this.checkCollision(this.x,this.y,this.width,this.height,"up",map) == 5){ // check for fire
      this.die("burning to death");
    }
    else if (this.checkCollision(this.x,this.y,this.width,this.height,"up",map) == 4){ // check for puzzle
      puzzleMap[level][stage] = 1;
      this.pieces += 1;
    }
    else if (this.checkCollision(this.x,this.y,this.width,this.height,"left",map) == 3){ // check for ladder on
    left side
      if (this.checkCollision(this.x,this.y,this.width,this.height,"up",map) != 1){
        this.move(0,-5);
        this.climbing = true;
      }
    }
    else if (this.checkCollision(this.x,this.y,this.width,this.height,"right",map) == 3 &&
    this.checkCollision(this.x,this.y,this.width,this.height,"up",map) == 0){ // check for ladder on right side
      this.move(0,-5);
      this.climbing = true;
    }
    else if (this.checkCollision(this.x,this.y,this.width,this.height,"up",map) == 0){
      this.jump();
    }
  }
```

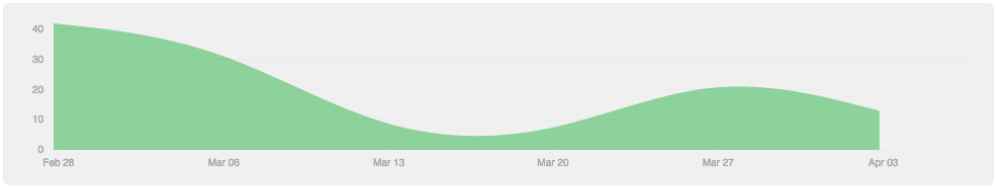
A few examples of the code I worked, including the Player checkCollision(), update() and draw() functions, and the renderPuzzle()function

```
draw(){
  var srcString = "resources/player/kirby";
  var needs_sprite = false;
  if (this.jumping === true || this.climbing === true){
    srcString += "Up";
  }
  else if (this.falling === true){
    srcString += "Down";
  }
  else if (this.running === true){
    srcString += "Run";
    needs_sprite = true;
  }
  else{
    srcString += "Normal";
    needs_sprite = true;
  }
  if (this.direction == "right"){
    srcString += "Right";
  }
  else{
    srcString += "Left";
  }
  this.image.src = srcString;
  context.drawImage(this.image, this.x,this.y);

  if (needs_sprite === true){
    srcString += "Spritesheet.png";
    this.image.src = srcString;
    context.drawImage(this.image,
    (Math.floor((this.anim_frame)/30)%(this.image.width/32))*32,0,32,32,this.x,this.y,32,32);
    this.anim_frame += 1;
  }
  else{
    srcString += ".png"
    this.image.src = srcString;
    context.drawImage(this.image, this.x,this.y);
```

```
else if (this.climbing === false){// falling code
  if (this.y + this.height + 5 > cheight){
    this.die("falling out of the world");
  }
  else if (this.checkCollision(this.x,this.y,this.width,this.height,"down",map) == 5){
    this.die("burning to death");
  }
  else if (this.checkCollision(this.x,this.y,this.width,this.height,"down",map) == 4){
    puzzleMap[level][stage] = 1;
    this.pieces += 1;
  }
  else if (this.checkCollision(this.x,this.y,this.width,this.height,"down",map) === 0){
    this.move(0,5);
    this.falling = true;
  }
  else{
    this.falling = false;
    this.jumping = false;
    this.jump_dist = 0;
  }
}
```

```
function renderPuzzle(puzzleMap,puzzleImage){
  for (i=0;i<9;i++){
    if (puzzleMap[level][i] == 1){
      context.drawImage(puzzleImage,i%3*50,Math.floor(i/3)*50,50,50,1150+i%3*25,75+Math.floor(i/3)*25,25,25);
    }
  }
}
```



This is a code contribution graph from GitHub; though my number is a bit inflated, about ~ 3000 lines of code is from bootstrap/jQuery.

Team Reflection

Overview

Sprint Cycle Backlog (Product Backlog)

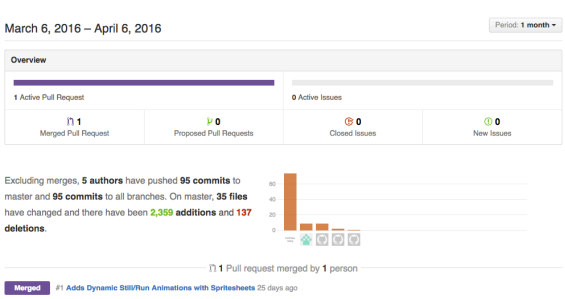
Matthew Wang edited this page 7 days ago · 12 revisions

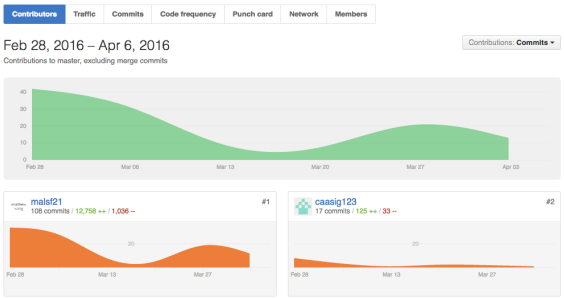
General Goals

Game Design

- Platformer
- Side-scrolling
- Has Levels
- Collect Puzzle to Advance to Next Level; Then, a full picture is made!
- Obstacles on the game map
- Enemies are also on the map, blocking your way
- Lives, after 3 lives you have to restart game. After each level, lives are replenished.

Going in, our team didn't seem to have high expectations. The game seemed quite hard, but with some work and research it might've been able to be completed. We had three members of our team completely unfamiliar with Javascript, as well as HTML, CSS, GitHub, and Object-Oriented Programming. As well, our designated graphic designer didn't know how to use Photoshop, and both of our designated coders hadn't used JS before.





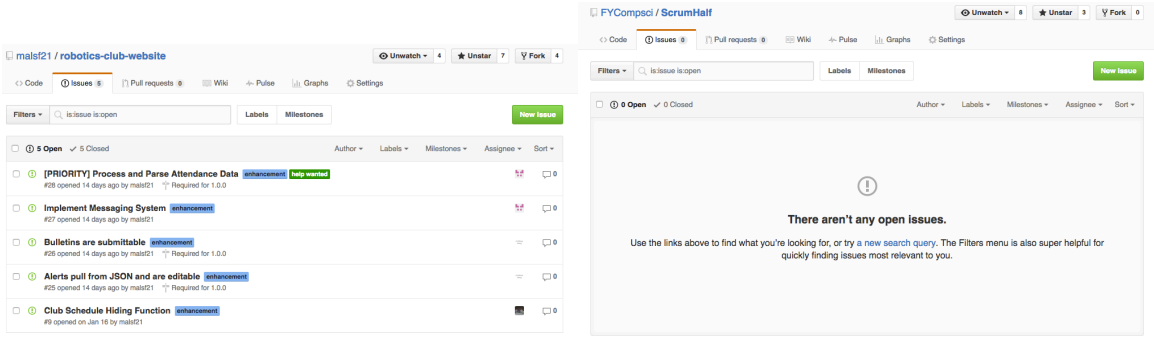
Our end product was both exceeding our expectations but also not meeting others. On the positive end, we ended up shipping a nice finished product: with movement, obstacles, gravity, lives, stages, the whole 9 yards. Saeed, our resource manager, in particular did quite well: he learned a few nice Photoshop skills and figured out how to make spritesheets! Unfortunately, our team coding designation didn't work as well as we anticipated: I ended up doing about 90% of the code in the game, and our designated coders ran into problems on their assigned tasks (the enemies). In particular, Aidan Heney ended up not being able to commit to the position, and didn't end up fulfilling any of his goals.

That being said, we worked together as a team to cover each other, especially as my job was to cover for any mistakes we had. Ultimately, we were pretty happy with what we had!

Start

If we could do this all over again, there would be a few more things I'd like to start doing:

- Get a tentative schedule on when developers are unavailable; that way, planning future goals would be easier
- Comment and layout code so it's easier for other users to interpret
- Create more modular code
- Standardize naming conventions (i.e. world1 vs. world_1 vs. world-1)
- Document issues using GitHub's Issue Tracker (shown below)
- Start harder parts of code first, i.e. Enemies



Here's an example of a good issue tracker, versus ours, which wasn't so great.

Stop

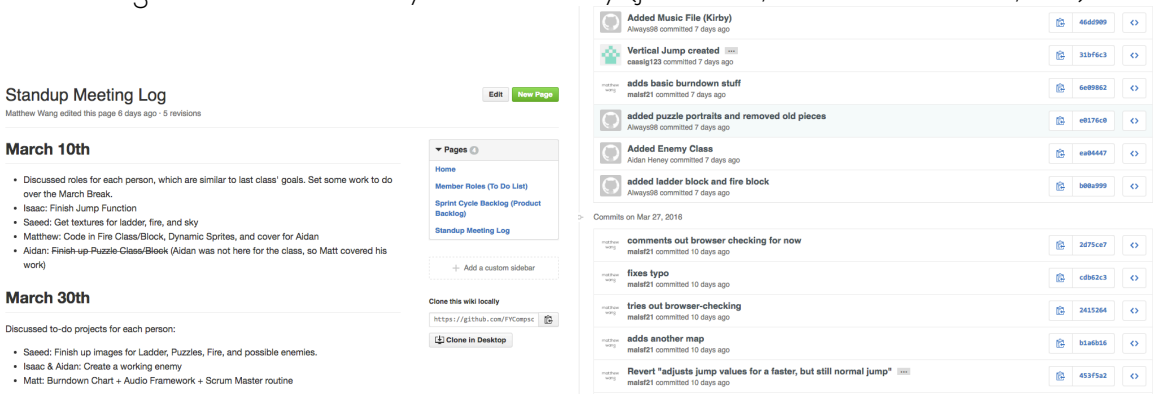
Likewise, if we could do this all over again, there are a few things that we might not want to do next time:

- Wandering off task during class time
- Creating functions and steps that are very hard to understand/very messy
- Not completing all of the certain goals per Scrum without notifying the team

Continue

That being said, there were some good practices that we should continue doing:

- Covering for each other if work is missed
- Communicating and asking for help if it's needed
- Using ECMA6 and OOP to make cool class-based programming
- Allow users to make their own maps/text
- Documenting our process vigorously through the Wiki, Website, and Burndown Chart (shown below)
- Organise files efficiently and intuitively (js folders, resources folders, etc.)



An example of the documenting and organisation that we did do.

That's it, folks!

I've got to say, this was my favourite class assignment we've had all year. We used GitHub, one of my favourite pieces of software, and used Javascript to make an pretty awesome game! In addition, I ended up learning a lot about Agile Scrum, and its artefacts and ceremonies. As a group, we learned a lot about Javascript, HTML, CSS, and how they interact; as well as how to work together as a team under the Agile Scrum methodology. Thanks for reading this monster of an assignment!