**Project Journal**

**Time Management:**

With this long term and complex program, time management was a crucial aspect of the project’s development. Monday, November 3rd marks the beginning of our work on our project. At this point in time, we were purely in the planning phase, not writing a single line of code for nearly a week. During the very first stages of our planning process, we decided what type of game we would focus on creating. Our team, *Print Stack Trace*, would focus on building a platformer game, which would include both sideways scrolling platformers (like Mario), and vertical scrolling platformers (like Doodle Jump).

After these initial meetings, we were individually prepared to work on our own time with periodic meetings (every Monday and Tuesday). I spent 4-6 hours per week (outside of our group meetings) coding for this project. My time spent on this project (on a weekly basis) breaks down in the following way:

* **4 hours coding new features**. Within these 4 hours, my productivity broke down in the following ways:
  + 30 minutes planning best design for new feature.
  + 1.5 hours reading documentation and tutorials in order to guide development of new feature.
  + 1 hour actually writing code.
  + 1 hour testing and debugging code.
* **1 hour refactoring.** For each new feature that I coded, I tried to spend about 1 hour afterwards looking over the code and seeing ways in which I could refactor to improve design / readability.

**Managing My Code:** Code management was an extremely important challenge in this project: how do you allow 10 people to be creative and work hard while making sure that our “master” branch isn’t get filled with merge conflicts or potentially dangerous code. Therefore, when coding a new feature or even fixing bugs, we did so in a **separate branch**, and then after pushing to that branch, we would make a **pull request** and assign it to one of our teammates who would review the code. This has a few advantages:

1. Bad / dangerous code is isolated on branches (not on master)
2. Having someone review your code helps catch dangers bugs
3. Having others review your code helps keep your team familiarized with different parts of the code base.
4. Responsibility for code is shared, so moving forward more people can work on the same areas of code even if they didn’t originally write it.

**Time Management Reflection:** Throughout this course, I have figured out ways to improve my efficiency. Planning out my **design of a new feature** was the most efficient way to spend my time, because this allows you to be even more focused, directed, and accurate when you actually begin to code, saving even more time. What often felt like a waste of time was attempting to understand someone else’s code without having them available for questions. In these cases, I could sometimes spend hours looking through the code base in order to discover what would end up being a very simple implementation that could have been explained in a few minutes if my teammate had been available. This obviously points to the need for more real time documentation so that even if your teammate is not available, you can easily understand what is going on in certain classes and features.

**Teamwork:**

Our team set reasonable expectation from the beginning, and worked very well together. Our initial planning meetings included the following discussions:

**Initial Planning Meeting (11/3 : 1 hour):** This meeting focused mainly on dividing the responsibilities of our project and tackled some logistical planning moving forward. In terms of planning we laid out the resources that we would likely need to be successful, including important links to the project guidelines as well as contact information for every group member. Furthermore, we set up a weekly team meeting time, which would end up being most Monday evenings at 10pm and most Tuesday evenings at 10pm. We expected that each team member would spend 75-100 hours working on this project.

**Initial Design Meeting (11/3 : 2 hours):** In this meeting, we focused on the high-level design of our project. We decided on our entry point into the game-authoring environments (PSTGameAuthor.java) and our entry point into the game-playing environment (PSTGamePlayer.java). We also began thinking about how our authoring environment would communicate with our playing environment – through the GameEngine.java class. We focused on discussing the important modules that would make up our authoring environment (front-end / back-end), our game engine, and our playing environment (front-end / back-end).

**Initial Sub-Group Design Meeting (11/ 9 : 1.5 hours):** After dividing responsibilities and specifying who would be working on which aspects of the program, Daniel, Petra and I met to speak about building the game-authoring environment. These meetings laid out the basic classes that we would need and discussed the design that we would pursue in building an extensible GUI for the game-authoring environment (factory-building of elements that use reflection).

Through these initial planning meetings, we were able to delegate responsibilities. Individual responsibilities broke down in the following ways:

* Game Data:
  + Ashwin
  + Marcus
* Front End (including both the GameAuthor and GamePlayer):
  + Petra – focus mainly on GameAuthor front-end, building the GUI framework.
  + Daniel – focus mainly on the GamePlayer front-end, however since we strove to build the game-authoring environment first, he focused first on building a decision table for handing collisions in the game-authoring environment.
  + Me (Nick Balkissoon) – My responsibilities were two-fold: I focused first mainly on working alongside Petra in building an extensible framework and implementing the front-end GUI for the game-authoring environment. After we had finished the game-authoring environment, I became the point person for GamePlayer, building the framework for that GUI. The second main aspect of my responsibilities was serving as a point of communication between the front-end and back-end. This required me to have an adequate understanding of both, in order to field questions and direct questions to the correct people. For example, when the front-end was having questions about how images were being stored and passed through to the front-end, I worked with Marcus and Ashwin to figure that out (passing image paths) and passed that information along to Daniel and Petra on the front-end side.
* Game Engine:
  + Pranava
  + Zach – the mastermind behind design. Zach did a great job clearly explaining good design and enforcing good design. It was important to have him in Game Engine which was the centerpiece of our program’s design.
  + Justin
* Author/Player Backend:
  + Ethan – Ethan’s role was similar to mine, serving as a point of communication between other groups and his. We worked very closely with this group, and therefore I worked with Ethan a lot. For example, we would tell them what sort of public method calls we would need in order to “create” objects in the authoring environment and have them passed along to the Game Engine, and Ethan was in charge of communication with us (front-end people) and working to make sure we had the correct API available to meet our needs.
  + Nick W.
  + Jack

**Communication**: Our team had great communication. We started a **GroupMe** (a group chatting app) that allowed us to all be in constant communication. We also created **Google Documents** where we could collaborate on creating our designs. Each sub-group had another form of communication. For example, Daniel, Petra and I communicated through **Facebook Chat**. By defining these lines of communication from the very beginning, we were able to keep people updated even when people were out of town with interviews and travel. Interviews ended up having an impact on our project. The majority of our team, at some point, had to leave Duke for a few days with interviews for jobs / internships. During these times they were obviously unable to commit, so this affected our productivity, however people were very good at being transparent about how much work they would / would not be able to complete while traveling.

**Project Plan Execution:**

* **Modular Development:** Our intention with our project plan was to modularly build the different parts of our program. This required us create APIs that would allow the other modules to interact with our module. This is a common project development plan and works very well in theory, however in practice we fell short in actually putting the pieces together. While we all implemented our various pieces very well, we were slow at actually putting them together, often waiting until the last minute which led to the discovery of problems that we hadn’t seen before when everything was separated.
* **Updates / Improvements:** Our project plan did not make it easy to update and improve code. With our current system of developing on a branch and merging with master only through a code review process, it was hassle to go through this process with only a few lines of changed code. Therefore, most testing and improving would happen on the branch before it was merged with master, and then once it was merged with master, we rarely went back and made improvements. One way in which I would suggest improving this process would be to have two accepted methods of pushing code to master. Aside from the method described above, we could have a “bug fixing mode” in which you are allowed to directly push small changes (less than 5 lines of changed code) to master directly without the need for a new branch and a code review from a peer.
* **Testing:** Our intention was to build up JUnit tests for all of our classes, but in practice we never were able to do this in a consistent manner. Although JUnit tests would have definitely contributed to less buggy code, I don’t believe that our code was particularly buggy because of a lack of tests. By abiding by our **branch 🡪 pull request 🡪 review** methodology, we were able to avoid a lot of careless bugs.
* **Team Roles:** Our team roles stayed consistent throughout the project. Because the code base got so large and complex, it was very difficult for someone to venture out of their module. The few times that I spent working in Game Engine ended up being extremely inefficient – it made more sense to explain your issue to someone within that sub-group and have them work on it. Therefore, we tended to stay within our *area of expertise*.

**Commits:**

We abided by a **branch 🡪 pull request 🡪 review** methodology, therefore we rarely, if ever, committed directly to master. The lifecycle of a new feature followed the following process:

1. Create a new branch off of master
2. Write code in new branch, commit, and push
3. When you are finished, create a pull request on GitHub
4. Pull request is reviewed by a team member
5. Pull request is merged into master by a team member

I committed a total of 9 times to this project. These commits averaged between 50 – 150 lines of code each, and occurred more towards later in the project. I don’t think that my commits accurately represent my contribution to the project because often my input was reflected in other people’s commits. For example, Daniel and I worked often very closely together in developing aspects of the Game Player front-end, but sometimes he would make a commit based on our collaboration and sometimes I would make a commit based on our collaboration. However, I think my commits and their commit messages were substantial and easily understandable. For example, the following commits:

* [**horizontal and vertical scrolling**](https://github.com/duke-compsci308-fall2014/voogasalad_PrintStackTrace/commit/55c94c6cd2dc1d17cbdc1b9760c79013d413d5d9)

commit 55c94c6cd2dc1d17cbdc1b9760c79013d413d5d9

* + This commit was a 125 line commit that implemented the ability for our game-authoring environment to scroll horizontally and vertically in order to build expansive game environments. My commit message adequately reflects this new feature.
  + This commit did not cause any merge conflicts as it was modularly built and not invasive on many classes.
  + This commit was done in a timely manner… since we review each other’s commits, we are constantly accountable to our teammates, so we try to do things as timely as possible.
* **cleaned GUI and pass Image path instead of Image**

commit b91fbdd452a606b0c6ea3c74bca7da0029fc419c

* + This commit was an 86 line commit commit that fixed a bug in our program in which we were trying to serialize Images with GSON when GSON and JSON cannot serialize images, so instead we had to pass the image paths and serialize those. I think that my commit message adequately reflects that improvement, however the “cleaned GUI” portion of this commit message is not very clear or specific. It would have been helpful to specify exactly which aspect of the GUI I had worked on.
  + This commit did not cause any merge conflicts.
  + I remember working on this commit at 3:00am so that our program could be fixed and ready to go for others to work on it when everyone woke up, so I would definitely say this commit was done in a timely manner.
* **GamePlayer can call GameEngine to get SpriteCharacteristics and LevelCharacteristics**
* commit 661d3790c4bda7f01671ec38920cd0b26b0d5ae1
  + This commit was a 25 line commit, but implemented very important methods in order to allow for our game-player front-end to begin rendering sprites. I implemented methods to get important objects SpriteCharacteristics and LevelCharacteristics, which would allow the front-end to render the game. I think that my commit message adequately representd the nature of this commit.
  + This commit did not cause any merge conflicts.
  + This commit was also done very late at night so that my teammates could take advantage of its functionality in the morning, so I would definitely say that this commit was done in a timely manner.

**Conclusions:**

I think our team was very efficient and effective. I think some of the keys to our success were outlining the process for communicating and having a great system for pushing new code and having it merged to master. I also think that we spent a great amount of time planning and preparing for the project, thinking purely about good design and how it might all fit together. However, I have the following areas of reflection:

* **Project Size Estimation:** I think we did a good job of estimating the size of this project in terms of its parts, but I don’t think we adequately budgeted for the amount of time it would take to get all of the pieces working together. This process, which included a lot of debugging, was an extra piece on top of all of the actual coding of new features which we did not prepare for. Additionally, some parts of our program were “easier said than done”. One of these areas in particular was creating a viewport for our game. This was something that we held off on until the last minute, and really ended up causing us a lot of difficulties. In the future, I think we should take a bit more time thinking through each part and thinking in depth about how it might theoretically be implemented. This could hopefully alert us as to which areas are more time consuming or easier than others. **All features are not created (or implemented) equally.**
* **Personal Contribution:** I think I was a solid contributing and dependable member of this team, however I wish I took up more of a leadership role. Unfortunately I missed over a week of school interviewing in CA, so I was unable to get as involved as I would have liked to. However, I think I did a great job of telling my teammates realistically what I was able / was not able to accomplish while traveling so that we kept reasonable expectations.
* **Editing Code:** One part of my code that required the most editing was rendering animations. I think that this was a particular difficult part of code to merge into our master branch because I was working remotely with the team while everyone else was working together rapidly. Therefore, it was difficult to talk through individual components that were not working together. Without this face to face interaction in working out the issues, this part of my code took a lot of editing and debugging to get it to fit together with the rest of the code (which was also rapidly changing).
* **Good Design:** This course has taught me a lot about good design, however there are definitely thinks that I should do to become an even better designer. Read. While in this class we have learned some great designs, such as Factory Design, Multi Methods, Visitor Patterns, Reflection, MVC etc., there are a lot of other great designs that people have come up with to tackle all sorts of problems. I should keep my eyes and ears open to these other designs and think about incorporating them in my programs. However, I think that the design that I have already learned in this course are a great foundation and can provide a great starting off point for any program. Another area in which I could improve is learning more about designs **that I already know**. While I have a basic understanding of the Factory Pattern, I should read more about it and see where I could improve.
* **Be a Better Teammate:** I think I have grown a lot as a teammate this semester. My communications skills have improved, I am more dependable, and I write code with other people in mind. However, I think I could still improve as a teammate by being more punctual. Because I live off campus, I was often arriving late to meetings, which delayed the entire group. I need to work on my punctuality.
* **Last Ditch Effort… Which Part Would I Work On??** I think that GamePlayer.java needs some serious refactoring. We used this class as sort of a “catchall” for everything in the game-player environment, and didn’t think too much about breaking it into smaller classes. Therefore, this class is a few hundred lines long, which is wayyyy to long for what it was originally intended to do (handle basic interaction between front and back end). I would break this up into more individual GUI elements and modular pieces that that handle the animations.

**Project Design**

**Status:**

**Code Style:** I think that our code is generally readable. We chose to follow camel-case naming conventions and “myVariable” naming conventions as well for instance variables, however this broke down in some cases. For example, looking in SpriteCharacteristics.java, we see that some instance variables are simply named:

private double width;

private double height;

instead of :

private double myWidth;

private double myHeight;

This is a clear break in our naming convention and definitely warrants refactoring. Additionally, from person to person, my teammates use different types of language and specificity to describe variables and methods. For example:

protected void addToOtherPane(Image image, String imagePath)

This method, “addToOtherPane” does not really give us any information about what, specifically, it is doing. We can infer we are add an Image to the “other” pane (based on its inputs), however what pane are we currently in? Which pane are we adding to? I would likely name this method:

protected void addLibraryImageToAuthoringPane(Image image, String imagePath)

These inconsistences in naming conventions and language exist throughout our program and are very frustrating when you are trying to read someone else’s code to determine what exactly a class or some methods are doing. A poorly named method, like the one just mentioned, would require comments to explain what it is doing, however, ideally, code should be written in a way that does not necessitate comments.