Helene

CS 3733 - D02

Professor Heineman 05-05-2015

Group: Nick Panzarino, Jeffrey Sirocki, Rupak Lamsal, Tyler Bennett, Jason Mckinney

Purpose

The purpose of this document is to provide evidence for the different roles of the team members. This report includes the effectiveness of our team in producing a finished software application during the 7-week term. Our report will describe of our project, team organization and responsibilities, operation process, tools we used, accomplishments, our reflection, and the lessons learned. This information will be used to reflect upon our group work and how we can improve our productivity when programming in groups for the future.

Introduction

Team Helene's Sixes Wild is a Candy Crush-esque game that includes multiple game modes where players remove numbered tiles by selecting tiles that add up to six. The application also features a robust level builder, which can be used for all game modes to define level parameters such as the layout and size of the board, the frequency that a certain tile number will appear at or the amount of moves/time that a player is allowed for the level. The application's usage of the Entity Boundary Controller paradigm allows for modular program design. This results in the ability to easily add new level types or other features easily, since the object/class hierarchy and code is simple and easy to understand.

Helene's five member development team is organized so that every member on the team participates in as much of each aspect of the product creation process as possible. This organization design has allowed the team to collaborate, share ideas and stay in the same state of thought while designing and implementing the application as well as effectively meet deadlines. In short, the vision of Sixes Wild was as much one member's on the team as another's as possible.

Team organization, members, and responsibilities

Team organization is essential to group productivity and success. Throughout the term we adapted our organization and team structure to produce software that is flexible, modular, and fully functional. The main keys to our success included organized meetings, team leaders, and discussion. At the end of the term, our finished product speaks for itself.

Our team approached meetings in a structured manner with purpose. We had meetings regularly on Mondays and Wednesdays for two hours each in a tech suite. By meeting in the tech suite we set professional expectations and a high level of importance for our discussions. Additionally, we met outside of these meetings for “work parties” in the zoo lab and “food events” to develop team chemistry and trust within the group at the goats head and boynton.

Early on, our meetings were heavily discussion based and focused on the modeling and analysis of our application. Our team was structured with the five roles described, but the roles that two roles that were truly exemplified were script and project manager. The project manager acted as the mediator of the discussion to keep everyone on topic in a time efficient manner and the scribe recorded a brief “minutes” of each meeting. These meetings adapted to include meeting agendas and discussion topics, another scribe responsibility. These agendas and discussion allowed increased productivity and allowed thorough modeling and analysis which helped set up our success later on. As deadlines approached we were in all on the same page and we all adapted to a more productive mindset where we split up the work to match strengths among team members.

Deadlines and implementation led to a logical strange in our team structure and organization. Our discussions turned into work parties. These work parties were generally longer and in the zoo lab. In these work parties a technical specialist would emerge and at the time would lead the group and delegate out tasks. This decision was usually made autonomously among the group and really helped manage the development of our application. Our implementation work parties for boundaries, controllers, testing, and documentation were all structured this way. As we neared the end of implementation we began developing a list of requirements that we needed to meet or milestones for the project. These further enforced our structure and helped with the delegation of tasks and productivity of the group. After work parties, members of the group would have individual tasks that they could work on their own on as well. This organization and structure during our implementation helped us meet deadlines in an efficient manner.

Overall, our team allocated time and resources efficiently over the term by adapting. The following shows the allocation of roles and responsibilities:

Week 1

Jeffrey Sirocki, Project Manager
Jason Mckinney, Release Manager
Nick Panzarino, Quality Assurance Manager
Rupak Lamsal, Technical Design Lead
Tyler Bennett, Scribe

Week 2

Tyler Bennett, Project Manager
Jeffrey Sirocki, Release Manager
Jason Mckinney, Quality Assurance Manager
Nick Panzarino, Technical Design Lead
Rupak Lamsal, Scribe

Week 3

Rupak Lamsal, Project Manager
Tyler Bennett, Release Manager
Jeffrey Sirocki, Quality Assurance Manager
Jason Mckinney, Technical Design Lead
Nick Panzarino, Scribe

Week 4

Jeffrey Sirocki, Project Manager, Release Manager, Modeling Specialist
Jason Mckinney, Quality Assurance Manager, GUI specialist
Nick Panzarino, Technical Design Lead, Modeling Specialist
Rupak Lamsal, Technical Design Lead, Modeling Specialist
Tyler Bennett, Scribe, GUI specialist

Week 5

Jeffrey Sirocki, Project Manager, Release Manager, GUI Specialist
Jason Mckinney, (Away with Robots)
Nick Panzarino, Entity Implementation Expert, Technical Design Lead, GUI Specialist
Rupak Lamsal, Technical Design Lead, Modeling Specialist, GUI Specialist
Tyler Bennett, Scribe, GUI specialist

Week 6

Jeffrey Sirocki: Project Manager, Technical Designer
Jason Mckinney: GUI Specialist, Technical Designer
Nick Panzarino: Technical Design Lead, Implementation Expert, Quality Assurance
Rupak Lamsal: Technical Designer, GUI Specialist
Tyler Bennett: Testing Expert, Technical Designer, GUI specialist

Week 7

Jeffrey Sirocki: Project Manager, Release Manager, Testing Expert
Jason Mckinney: GUI Specialist, Technical Designer, Testng Expert
Nick Panzarino: Technical Design Lead, Implementation Expert, Quality Assurance
Rupak Lamsal: Technical Designer, Documentation Expert
Tyler Bennett: Testing Expert Lead, Documentation Expert

Process

Our application development process adapted throughout the term. During the design and analysis stages we valued organization and discussion. We spent a four hours a week discussing design implementations in a tech suite and two hours outside of the tech suite. This process evolved over the term and meeting times were adjusted. Overall, we met often and regularly and these meetings allowed us to thoroughly go over our design, model, and implementation strategy before we began programming.

Our implementation process adapted as we went as well. Our meeting schedules got more irregular and longer to be more productive. We first implemented the entities and tested them. We then incorporated the boundary objects made in WindowBuilder and linked them to

the entities. After we implemented controllers to get the use case functionality. Lastly, we added undo/redo capabilities and testing.

Behind every unit in our code was the idea of modularity and flexibility. Classes that were written by one person could be interfaced with classes written by another. They were also designed to be flexible and not break if a related class was modified.

Our team followed the modeling and analysis process that was also implemented in the individual assignments. We began by discussing all the possible user stories associated with the Sixes Wild game, breaking them into two groups- a builder group and a player group. From there, we created our use cases. After much discussion and evaluation, we crafted the use cases that became the basis for our controllers. Our group then modeled the relationship between all of our potential entities, boundaries, and controllers through the creation of a UML diagram.

From that point, we began coding. To meet our objectives, we split up the code into pieces and group members tried to implement the functionality that was assigned to them. If a part of the code could not be figured out, team members helped one another out through pair programming during group meetings.

Through our frequent group meetings, we were able to effectively communicate which parts of our code needed to be finished to meet our requirements in a timely fashion. Whenever there was a deliverable due, our group meet in the days leading up to that due date and reviewed our work as a whole while constantly reminding ourselves of the criteria of the deliverable. Through this process, we were able to ensure that our code not only met deliverable requirements, but met the standards our team set out to achieve as well.

Communication was key to our success. Our team used Facebook Messenger, text messaging, and outlook meeting requests for communication. This was the most convenient way to do it, every member had access to these methods, and we thought it was best for the team.

Tools

The tools that we used to complete the modeling and analysis included google drive/docs for documentation, reports and deliverables, StarUML to design the UML diagrams for the structure and design of the application, and WindowBuilder for the design of the GUI. Google's collaboration suite offers a simple enough set of tools that the team has no complaints on. StarUML was valuable for designing the application models, and the design and the finished look of the diagrams is slick, but the user interactions with the application are somewhat clunky and slow. WindowBuilder was easy to use and had enough simple tools to layout our application into a presentable view. The team would likely use both StarUML, google docs, and WindowBuilder again.

Accomplishments

As this was an instructional assignment, there was a list of requirements which we were required to fill. We are happy to say that we successfully completed all the requirements, which are listed below:

Status	Description
Done	Able to run builder
Done	Able to create new level
Done	Able to load existing level to be edited
Done	Able to change level parameters
Done	Able to change level type
Done	Able to mark individual squares as being inert
Done	Able to preview level to see what it might look like
Done	Able to save level
Done	Able to undo
Done	Able to redo

Status	Description
Done	Can run game player
Done	Can select unlocked level to play
Done	Stores persistently when level is won, its stars, and highscores
Done	Can unlock new levels based on completion of existing ones
Done	Can generate new board given level parameters
Done	Can make a move and remove tiles
Done	Can demonstrate special moves(Swap, Remove, Shuffle)
Done	Can demonstrate gravity
Done	Show countdown timer
Done	Show level stops once timer expires
Done	Show which square have yet to be eliminated
Done	Once moves are made, demonstrate squares become eliminated
Done	Show level stops once all squares have become eliminated
Done	Show a 6 falling into a release bucket
Done	Show level completing once all buckets are filled

In addition to the standard requirements, we were able to implement a number of unique features which added greatly to the quality of our application.

1. **Gravity Control:** In our game, Gravity is defined by a “Fill From” reference that each square on the board maintains to another square or source. This made it easy to modify the behavior of gravity in our game. After making some minor changes/additions to some of the entity objects, and creating a GUI, and one controller, we were able to add a powerful, easy to use interface for a user to change way tiles fill on the board to be anything they could imagine.

2. **Modular design:** Although this feature is not very visible to the user, it is a very powerful one which made up the core of our project. Our application was designed to be easily expandable. Classes are designed like functional blocks that can be interchanged to modify behavior. For example, moves are created from an AbstractMove class. A move can define for itself any behavior it needs, but as long as it extends AbstractMove, both the Builder and Player will be able to perform the appropriate actions involving the new move, without the user having to modify existing code. Similar functionality exists for Levels, Square types, Frequency Lists, and Game modes. We made heavy use of Inheritance, Abstract classes, where appropriate, Java Reflection to achieve this result.

3. **Flexible GUIs:** All of the GUI's we created were designed to be completely resizable. The board, sliders, panels, fields and labels are arranged so that they are automatically displayed properly, even if the user drags or moves the window. Furthermore, in some specialized Boundary objects, fields and components are added dynamically. For example, frequency sliders and text boxes are added based on the number and value of frequencies in a FrequencyList object. Spawn frequencies, which range 1-6, will automatically display 6 sliders and text boxes, and automatically assign controllers. Bonus frequencies, which range only 1-3, will only add three sliders. Additional functional panels can be added to the "options" combo box, and it will automatically be made active and visible when the user selects it .From a development standpoint, this was useful because the exact position and size of components did not to be firmly decided upon before implementation. We could resize or move functional panels and components without having to worry about fiends not displaying properly. We could also add fields or modify entity objects, and the GUI would update accordingly while remaining functional.

Our team managed to complete all the required tasks, as well as every additional feature we wanted to have. If we had more time we might have liked to smooth/clean up some parts of the interface, possibly add a game over screen, or implement achievement tracking, but we were content with the features we already had and decided to focus more on testing and refining what we had than on adding features.

Deliverables

1. SixesWild Player Aplication
2. SixesWild Builder Application
3. LevelData file with 16 pre-made levels.

Reflection

What worked, what didn't work

Our regular meetings during the first phase of the project helped us build a strong foundation to begin with. We were really organized and everyone was on the same page. However, as we reached to the implementation phase, organizing things became much more difficult because there were so many things to be done. We did make todo lists and tried to finish jobs in an organized manner, but it wasn't as seamless as we thought it would be.

Our biggest mistake

Our biggest mistake was ignoring undo and redo until the last few weeks. We focused more on getting the game to work first, and just completely didn't think about undo/redo while we were at it. Later when we started adding undo/redo to our code, we realised how easy it could have been if we had thought about it right from the beginning.

Changes we would make

It's important to be aware about all the elements in the projects right from the beginning. Ignoring small things that seem insignificant at the beginning might turn out to play a major role towards the end. Be organized, think through and make sure that the entire team is on the same page before starting to code.

Lessons learned

Starting small is the best way to go. Get the framework done first. Think of as many things as you can and make methods for them right at the beginning so that it will make things much easier and smoother while you start writing the main game.

Things we learned

It is difficult to remain organized throughout the project especially towards the end of the term when there are so many things to be done and so little time. The best way to cope with this problem is to start early, meet regularly and think ahead as much as you can. Our team had started writing a few basic foundation classes as soon as we were done with the design phase.

There is no time to relax and take a break until you are actually done. Don’t be proud that you got the design phase done, and take a break before starting the implementation phases. Get right at it. Spend at least a couple of hours every day. Even though it seems like there is nothing to be done for the next day, just meet up and start thinking about things that can be done in the future. The more you talk about this stuff, the easier it gets.

Advice to future teams

Go to class. Do the work. Ask for help. Don’t try to spend too much time on things that are not important. Stay focused and do what the professor wants you to do. If you don’t know how to do something, take the time to figure it out properly and do it right rather than finding a workaround. If you keep avoiding problems and going around them, then they just pile up and the project becomes an unstable mess.