Individual Report – Molly Henry (300562038)

I oversaw the App module for the project and so I created the main window for the game, took care of user input and adapted information between other modules.

Substantial/Important Contributions	
Commit-URLS	**Description**
2d199c82	Initial window and menu bar of game, the Base of all the rest of my work which the rest of my team needed to begin working with
ec0bfe51	Added different Controller states for different windows, which allowed all the different Key strokes to be allowed at different times
d45d56b6	Added classes to override JComponents and therefore condensing my code down substantially and making it easier to unify the look of the UI

Group Rankings		
Name	**Email**	**Score**
Molly Henry	henrymoll@ecs.vuw.ac.nz	5
Chris Sa	sachri@ecs.vuw.ac.nz	5
Gideon Wilkins	wilkingide@ecs.vuw.ac.nz	5
Diana Batoon	batoondian@ecs.vuw.ac.nz	5
Abdulrahman Asfari	asfariabdu@ecs.vuw.ac.nz	5
Gavin Lim	limgavi@ecs.vuw.ac.nz	5

Our whole team worked well together and pulled their weight in each of their modules. We communicated well and were able to easily collaborate on parts of the project where our modules interacted together. Therefore, I have scored us all 5/5.

Reflection Essay:

Overall, I am very pleased with my project, the things I learned, and how well our team worked together. There are some general things I would like to note about my work before I answer the questions below. When we began the project, I was unaware we had an @ecs.vuw.ac.nz email address or that that was the one GitLab used to commit things with. This is why the first half of my total commits are with my @myvuw.ac.nz address. Another thing to note about our group's commits is that until approximately integration day (most of us) were squashing commits before merging branches into main. We thought it would keep our Git log tidier, but we later released we wouldn't have enough commits to show as evidence for this assessment.

What Knowledge and/or Experience have I gained?
I am now much more proficient with using Git, especially with GitLab. I can create, assign, and label issues in GitLab for better task management within a project, as well as using issues to create branches and merge requests for resolving said issues. I have gained experience working within a team and now understand better how delegating modules is a safe way of splitting up tasks between members of a large project and a good way to organise code.

What were the Major Challenges, and How did we Solve Them?
One of our first major challenges was getting our project to run on the ECS machines. We started testing this a week before our scheduled integration day, and still couldn't get it working fluidly enough to be able to run it with breakpoints on the day. Our issue was with Gradle and the mismatch between both Java and Gradle versions on the ECS machines and our own. After our

integration day we decided our final hand-in would be smoother if we removed Gradle all together. So, we restructured our project, manually imported a couple libraries, and removed Gradle.

Our second major challenge was with the Recorder module. From the beginning we knew we wanted to be able to play our recordings forwards and backwards. Which led to some issues in how we were going to record moves to make sure all moves would be reversable. Over the past 6 weeks we tried 3 or 4 completely different Recording designs before we found one we felt would work. Our last design still had kinks to work out, but it now runs very smoothly, and is rather robust. It took a lot of teamwork between App, Domain, and Recorder to be able to solve this challenge, but we manged to achieve what we wanted. Our final hand-in is unfortunately still not perfect, as Recorder produces unexpected results when trying to replay a game that was started halfway through (via Persistency loading). We had a plan on how we would solve this but ran out of time to implement on our final day.

Which technologies and methods worked for me and the Team, and Which didn't and Why?

GitLab issues and worked well for our team. We assigned them to each team member, tagged with priorities and respective modules and created all our merge requests from them. We made sure to assign team members to their relevant issues and assigned reviewers on the merge requests to get approval and feedback before merging into Main. Requiring approval before merging was a good strategy for keeping us all aware of the changes being made within the program. Issues in general kept us focused on the right tasks at the right time without forgetting anything. Another Git helper was an app called GitKraken, it allowed us to visualise the different branches and commits in our program and made committing, merging, switching branches and rebasing easier.

Technologies that didn't work too well would be Gradle as previously discussed. It caused compatibility issues between the ECS machines and our own.

A method that worked for our team was pair-programming. I believe our team communicated excellently and this allowed pair-programming to work nicely. When working on parts of the project that interacted with other modules or on a tricky bug we would screen-share to each other in Discord and assist in the coding process. This also meant that all our team got to learn each other's modules quite well and allowed for faster bug fixing later down the road.

Discuss how you used one particular design pattern or code contract in your module. What were the pros and cons of using the design pattern or contract in the context of the project?

My App module uses the Adapter pattern first and foremost. As App can talk to all other modules it exists as a bridge between all the modules that could not directly interact. The most prominent example is between Recorder and Domain. Our Recorder needed to know about the state of Domain after every action but could not get the information from Domain itself, so we used the App module to adapt the information from Domain and package it and pass it Recorder. The pros of this pattern were that App didn't need to concern itself with what the program was *actually* doing at any given time. It could just pass the information to the appropriate places for the other modules to process. The cons might be that this gave the App module a lot to do every game tick, and potentially reduced the speed of our program.

My entire App module was also the Controller aspect of the Model-View-Controller design pattern. I took in the user input via buttons and key presses and then controlled other modules into doing the related actions. The pros here is that other modules don't need to worry about user input and only need to do as App tells them when App tells them.

What would I do differently if I had to do this project again?
I would try to use more static methods within the base of my program. This would, hopefully, reduce the number of times I pass unsafe instances of my Base class to other classes as I would be able call the static actions instead of calling them on the instance of Base.

I would also like to have a better class and method name style established quicker, to reduce the amount of refactoring that happens as the project progresses. The constant refactoring and shuffling of methods occasionally made Git merges a headache and caused some confusion for my other team members.

What should the team do differently if we had to do this project again?
We would use more GitLab issues with attached merge requests from the issues from the beginning of our project for extremely organized tracking of the project. At the beginning of the project, we mainly worked from module specific branches that we would occasionally rebase onto main or merge into main. As the project continued and the amount of code grew, rebasing got trickier, and merges were happening more frequently and so we switched to creating branches from the issues we were trying to resolve and instantly it was a lot easier to quickly resolve the issue and merge it into main. On a related note, we would also rebase and merge to main more quickly in general.

If we did this project again, our team also would've made a (more extensive) Util package for storing project wide information. We discussed having one to store the Direction enum, and UI colours in towards the beginning of our project but thought it would be against the brief so declined to do anything. This means when Recorder is storing information about the player's Direction, we store it as a String instead of Domain's enum. We were not a fan of this approach but couldn't see a way around it. It wasn't until the final days leading up to the hand-in that we realised we would've been allowed to add a module like this but at this point it would've been too much work to refactor all the modules.