Development Plan
SFWRENG 4G06
GitHub

Team 9, dice_devs
John Popovici
Nigel Moses
Naishan Guo
Hemraj Bhatt
Isaac Giles

Table 1: Revision History

Date	**Developer(s)**	**Change**
2024-09-19	Hemraj Bhatt	Added content to sections 7, 8 and the appendix. Created reference page
2024-09-20	Nigel Moses, Naishan Guo	Added content to sections 5 and 6.
2024-09-23	Isaac Giles	Added content to sections 1-4
2024-10-24	John Popovici	Added clearer link to GitHub
2024-10-24	John Popovici	Updated team member roles
2024-11-05	John Popovici	Refined team meeting and communication plans
2024-11-05	John Popovici	Refined team charter and rules
2024-11-05	John Popovici	Added to workflow plan
...	...	...

1 Team Meeting Plan

Team will meet every Friday morning at 9am via Microsoft Teams meeting for 30-60 minutes for a weekly meeting. These meetings will discuss all upcoming deadlines within the next 2 weeks and feedback from previous peer and TA reviews. When a deadline is to arrive, the workload will be discussed and divided amongst the team as they are best suited to fulfill the criteria of that section. Team members should arrive with additional things they wish to discuss as well, as this meeting should have all members and will be an efficient way to quickly communicate all matters.
Team roles within the meetings will be as described in the subsection below about team member roles and the meeting will be summarized and posted as an issue on github.
Meetings with the supervisor and TA will occur as would best fit the timeline of the project and at important milestones such as a new deliverable for TA meetings, or a proof of concept for the supervisor. Further meetings will be suggested in Microsoft Teams chat when team members feel that an additional meeting is needed.

2 Team Communication Plan

Team will communicate via Microsoft Teams messaging for all baseline communication needs. We will also use Microsoft Teams to setup team calls for both recurring meetings and spontaneous meetings that come up when needed.
Communication with TAs and supervisors will be through emails and primarily carried out by the team liaison, as per the team member roles subsection below. Issues will be created for all meetings and lectures, as well as used to track feedback and our work to resolve and use said feedback. They will also be used for bugs we need to fix or features we are looking to add once the project has taken a more complete form.

3 Team Member Roles

All team members will be responsible for attending all meetings (lectures, TA meeting, team meeting) and for the development of both the documentation and of the game itself. All are expected to contribute to the team, follow the development workflow, partake in code reviews, and contribute as full stack developers. To better organize our team and responsibilities, we can highlight some key responsibilities each member will take on.

- John Popovici
  - Team Liaison: Responsible for communication between the team and external interests including supervisor(s), other capstone teams, and teaching assistants.

  - Meeting Chair: Leads team meetings and notes relevant information for bookkeeping.
  - LaTeX Specialist: Will leverage LaTeX to best assist the team such as setting up files for documentation.
  - Repository Admin: Will oversee GitHub usage and ensure usage of issues and commits.
  - Game Theory Analyst: Responsible for theoretical game model and probability calculations.

- Nigel Moses
  - Development Lead: Will take charge of development through GoDot and distribute responsibilities to those working with him.
  - Game Designer: Responsible for overseeing game design and direction.
  - UI/UX Developer: Will develop the user experience for a unified and accessible experience.

- Nai Shan Guo
  - Game Logic Programmer: Responsible for implementation of core game mechanics and handling turn-based gameplay with flexibility and modularity for variants.
  - Public Testing Lead: Responsible for spearheading and communicating to external testers for game feedback and demos.

- Isaac Giles
  - Development QA: Responsible for testing and tracking bugs and issues related to development. Will bring into the main development branch the most stable versions before further development is done.
  - Documentation Lead: Will oversee the writing of and maintenance of required documentation items for the purposes of the capstone project and the uploading of final documentation for deadlines.

- Hemraj Bhatt
  - Network Specialist: Responsible for the implementation and testing of multiplayer functionality and managing game state synchronization.
  - Gameplay QA: Responsible for testing general gameplay features and ensuring stability.

4 Team Charter - Ground Rules/Expectations

The following are the rules all team members agree to follow. These rules cover a wide range of topics including but not limited to quality of work, frequency and duration of meetings, as well as proper format for contact outside of scheduled times:

- Each team meeting must include an agenda that involves every team member. Involvement is defined as either assigning new work to someone or having that person update the group on the status of previously assigned tasks. This update will be prepared by each team member for themselves prior to each meeting, when presented.
- Each meeting will have a designated meeting lead responsible for guiding the discussion and keeping the team on track. The meeting lead is as per the team roles subsection, but if missing would be replaced by the next person in the team members role list in descending order.
- Scheduled meeting times are defined as tutorials and the weekly 9:00 am Friday meeting. If members need to schedule an additional meeting outside of scheduled meeting times, all members must be notified at least 24 hours before the proposed meeting time.
- During weekly meetings, the work that each member will be responsible for until the next meeting and/or deadline must be established.
- All team members are expected to attend all team meetings and at least one team member shall be in attendance for every lecture and every tutorial.
- All team members will use the Microsoft Teams chat for facilitating meetings, calls, and general communication. Additionally, members are expected to check their Teams messages at least once a day. If urgent contact of a team member is required, members are permitted to contact each other by phone.
- All major work requiring a merge request will be reviewed by at least 2 other group members before merged. This includes large code changes requiring multiple commits in a separate branch or documentation that needs to be reviewed and understood by all. All work submitted for a deadline must be reviewed by all team members and agreement must be confirmed on the Microsoft Teams chat by all members before submission deadline. Furthermore, work must be distributed at the latest during the weekly meeting before the deadline to allow time for completion, and all team members must attempt to complete their assigned work at least two days before the deadline to allow time for review and revisions.

5 Workflow Plan

The general workflow for the team follows from team meetings where we will assess where we are in our progress on the project, at which point we will determine what items need to be worked on next and work will be divided up accordingly.

Each individual team member will work on changes in their own branch for large multi-commit changes to the codebase and important documentation changes that need to be reviewd, and create a pull/merge request when ready to merge. Pull requests that are being merged into main must be created with a descriptive title and description. Smaller changes that are self-contained within a single commit and do not require testing, such as for documentation, can be pushed to main unless a specific commit pattern was decided upon ahead of time.

Commits are to start with a specific emoji as per our contributing file followed by a short description, such that they are readable and clear in their purpose.

Additionally test cases will be run automatically with GitHub Actions and any failed test cases will block merging. If all test cases pass, pull requests will still require approval from another team member that has reviewed the branch, to be merged into main. It will be expected that all of these steps are followed by all team members throughout the course of the project.

Project deliverables will discussed and tracked in weekly meetings and posted using GitHub issues. Peer review and TA feedback will also be tracked using issues with respective commits fixing those issues closing them. Once a stable project is put into place, feature requests as well as bugs will be tracked via issues as well. Templates for all of the above items are within the repository and will be used as appropriate.

The git repository used for the purposes of this project will be hosted on GitHub at this link here. The use of GoDot will not have a major effect on the use of GitHub outside of requiring the setup of additional directories and file extensions in the .gitignore file. All team members will have GoDot set up with the same configuration such that the project being maintained on GitHub can be opened, accessed, and modified by all. Code and assets will all be committed within the repository.

In Summary:

Group Steps

1. Discuss current status of the project.
2. Divide workload evenly.

Iindividual Steps

3. Pull changes from main branch.

4. (Optional; Recommended) Create new branch based on most up to date main branch.

5. Carry out task dev work and testing as needed.

6. Commit code with descriptive title including necessary emoji to imply the general purpose of the commit, and also add a descriptive commit description.

    - Automated unit tests will be run. If any tests fail user must recommit to fix the issue.

7. Other team members review and approve the merge request if no issues are noted.

    - If reviewers find an issue they can request code changes at which point the developer must go back to step 6 and make a new commit to meet code review expectations.

8. Merge code from main into user branch if user branch is out of date.

9. Merge code into main branch.

6 Proof of Concept Demonstration Plan

Developing an online multiplayer variant of Yahtzee presents several risks that could impact the success of the project. One significant risk is the inability to implement online multiplayer functionality with short response times, which is crucial for maintaining a smooth and enjoyable user experience, and to allow for simultaneous dice rolls. Delays or lags in response times could frustrate players and negatively affect gameplay, and hinder the player interaction that the game aims to provide. Another risk is not being able to correctly implement the complex Yahtzee scoring system, which could result in incorrect outcomes or unfair gameplay. Ensuring that the game accurately tracks scores according to the traditional rules is essential for user satisfaction. Finally, failing to implement the basic Yahtzee mechanics, such as rolling dice, re-rolling, and selecting scoring categories, could hinder the core gameplay. If these foundational elements are not properly developed, the entire game experience could become dysfunctional and unplayable.

To address the key risks in developing an online multiplayer Yahtzee variant, the proof of concept demo will focus on demonstrating the core functionality required to overcome these challenges. First, it should showcase the implementation of online multiplayer with fast response times, ensuring real-time interaction between players without noticeable lag. This will prove that the game can handle smooth, synchronized gameplay across different devices. Next, the demo must include a fully functional Yahtzee scoring system, correctly calculating scores for all possible combinations, like full house, straights, and Yahtzees, to show accuracy and reliability in scoring. Lastly, the demo should feature the basic Yahtzee mechanics, such as rolling dice, re-rolling up to two times, and selecting score categories, all working seamlessly. By focusing on these critical features, the demo will provide confidence that the project can overcome the identified risks and deliver a functional and enjoyable multiplayer experience.

7 Expected Technology

For this project, the online multiplayer variant of Yahtzee will be developed using the Godot game engine with GDScript as the primary programming language. GDScript is specifically designed for Godot and offers an easy-to-read, Python-like syntax, making it ideal for rapid game development and ensuring seamless integration with the Godot engine's core features. The Godot engine will provide the necessary framework for game physics, rendering, input handling, and networking, allowing us to focus on game-specific functionality such as the multiplayer interaction, Yahtzee mechanics, and scoring system.

External libraries may be utilized to support features such as real-time multiplayer synchronization and latency management. For example, Godot has

built-in networking features, but depending on the complexity and performance requirements of the game, we may need to integrate additional libraries or APIs to enhance real-time communication between clients. As part of the development, we will implement certain major components ourselves, such as the Yahtzee scoring system and game logic, despite the existence of general-purpose libraries. This is because the game mechanics are fairly specific and require careful tuning to ensure proper scoring and gameplay flow, which can be more effectively managed through custom implementations.

For this project, Continuous Integration (CI) will primarily focus on documentation updates, ensuring that the project documentation is always up to date with the latest development changes. This will help streamline collaboration between team members and provide a clear and consistent understanding of the project's current status. Documentation CI pipelines will automatically trigger when changes are made to files related to the design, development, or architecture, ensuring that our design documents, requirements, and user manuals are consistently aligned with the latest codebase.

Additionally, although CI can be more challenging in game development due to the graphical and interactive nature of the project, we will explore ways to incorporate CI practices into our development workflow. For example, we plan to investigate the feasibility of automated unit testing for core game logic, such as the Yahtzee mechanics, scoring system, and multiplayer interactions. While automated tests for graphical interfaces and real-time multiplayer can be difficult, CI could still help us ensure that the game logic functions correctly after every code change. By keeping CI lightweight and focused on these areas, we can ensure some level of automation without over-complicating the process, especially given the unique constraints of game development projects.

Performance measuring tools like Godot's built-in profiler will be used to monitor frame rates and resource consumption, critical for ensuring smooth, responsive multiplayer gameplay. Lastly, we will be using version control through Git, integrated with platforms like GitHub for collaboration, which will facilitate efficient management of code changes and team contributions.

8 Coding Standard

The coding guidelines below will allow for a uniform codebase and proper collaboration between developers. The guidelines are derived from Microsoft common C# code conventions [1] which are universally accepted as industry standards.

- Use "int" rather than unsigned types. `int` is commonly used in C# and interacts more easily with other libraries. Exceptions apply only for documentation specific to unsigned data types. [1]

- Class and method names should always be in Pascal Case
  - Ex: PascalCase
- Method arguments and local variables should always be in Camel Case
  - Ex: camelCase
- Employ abstraction, encapsulation, inheritance and polymorphism
  - Abstraction: Model the relevant attributes and interactions of entities as classes to define an abstract representation of a system. Use interfaces or abstract classes to provide a blueprint for other classes without revealing implementation details. [2]
  - Encapsulation: Hide the internal state and functionality of an object, allowing access only through a public set of functions. Use private fields to hide an object's internal state. Provide public properties or methods (get and set) to access or modify the private fields in a controlled way, ensuring safe interaction with the object. [2]
  - Inheritance: Create new abstractions based on existing abstractions. Reuse common functionality from the base class while allowing the derived class to add or override methods or properties. [2]
  - Polymorphism: Implement inherited properties or methods in different ways across multiple abstractions. Use interfaces or abstract classes to define common methods that can be implemented differently in each class. [2]
- Use a reputable linter; all members should use the same linter
- Write code with clarity and simplicity in mind
- Avoid overly complex and convoluted code logic
- Comments should be consice and to the point
- Code should be formatted appropriately
- Hard coded constants should be avaided, employ symbolic names and use configuration files
- Any code taken from external sources must be sufficiently cited through comments
  - Include links to webpages for any external resources
  - Include input and output for any AI-aided code

9 Project Scheduling

The team has decided to employ a Gantt chart to schedule the project. A Gantt chart provides a way to depict the duration, completion percentage, and leads of tasks in a visual manner. Due to these benefits, deadlines are met due to the duration of a task being enforced by start and end dates. It also allows for effective communication as others know who to refer to if there is an issue pertaining to a certain task due to the leads being listed. All in all, a Gantt chart provides great facilities to effectively plan out a project.

The team will also implement a backlog for our development which will be featured in our readme file on github to aid potential contributors. This will allow them to understand which features have been implemented and which are still in development. This will be in addition to utilizing GitHub project boards.

Gantt Chart: Link

Appendix — Reflection

1. **Why is it important to create a development plan prior to starting the project?**

   A development plan is essential when taking on any sort of project. It allows for the team to set a clear set of goals and milestones while also defining the project's scope. It creates a baseline of expectations for each team member and reduces any confusion about the team's plan as a high-level plan is detailed within the development plan. Without a development plan, a direction for the team is not set which in both the short and long term will cause issues that will eventually lead to missed deadlines and poor quality of work.

2. **In your opinion, what are the advantages and disadvantages of using CI/CD?**

   In our scenario, the team is creating a game using Godot, an open-source game engine. Therefore, there are some issues that present themselves. One is, performing automated testing, as it is more complex due to the need for graphical tests and user input simulations, which aren't easily automated in standard CI/CD pipelines. Another issue with CI/CD is that they are complex to set up and maintain as the system must remain updated, secure, and functional.

   However, there are still benefits to employing CI/CI pipelines as it promotes collaboration. Everyone is always working on the most up-to-date version which reduces commit conflicts and new features not working due to code logic being altered in an unpulled commit. It also allows new features and fixes to be pushed to production more quickly and reliably which enhances software quality and user satisfaction.

   Overall, implementing Continuous Integration (CI) and Continuous Deployment does provide a sizable benefit to most projects. However, as mentioned above, it cannot be used in certain circumstances. It may be ideal but it is not feasible. In the team's case, CI/CD may not be beneficial and impractical but we can still use aspects of CI/CD in our methodology.

3. **What disagreements did your group have in this deliverable, if any, and how did you resolve them?**

   When constructing the development plan, the team faced no disagreements. Due to the area of the project being outside of our expertise, the team was open to suggestions on how the project could be developed. Due to close to zero experience in game development, when researching important tools that would be used to develop the game such as what game engine to use we were able to go off separately on our own and research the pros and cons of certain game engines such as Unity and Godot. Doing this allowed each member to understand the pros and cons of different game engines and we were then able to conclude which game engine to use

without much issue. This was the way the team approached any aspect of the deliverable, which ultimately led to the completion of the deliverable without any issues.

Appendix — References

[1] B. Wagner, "C# Coding Conventions," *learn.microsoft.com*, Available: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions.
[2] B. Wagner, "Object-Oriented Programming C#," *learn.microsoft.com*, Available: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop.