Project Title: *Stalingrad: 1943*

By Jaden Pereira

Executive Summary

Stalingrad: 1943 is an arcade-style game, similar to classic games such as *Midway: 1942*, where the player controls a fighter plane and has to fend off enemies in the air in World War 2 themed battles. The game includes a mouse-based player movement system, a click to fire weaponry system, randomly generated enemies, and a scoring system based off of how many enemies are destroyed before the player loses all their health points, all built using PyGame. The project itself will include both planning, training and development phases. Game planning and worker training will take 9 days, and the actual game development will take another 9 days. One person will be working on this project with a budget of $600. Risks include time constraints, complex designing, difficulties involved in integrating various game mechanisms together and higher performance requirements.

Project Description

The goal of this project is to create a classic 2D arcade style shooter game, based on World War 2 and previous arcade games of the same theme such as *Midway: 1942*. This game, *Stalingrad: 1943*, will be themed after the Battle of Stalingrad, with the player controlling a Soviet Yak-3 and being tasked with fending off waves of German fighters and bombers. The game will be very easy to play as the controls are extremely simple, with the player only needing their mouse to play the game. The player's plane tracks the cursor's position, meaning movement is seamless and easier than a joystick or key-based system. Shooting also relies on the mouse, with any mouse input (left, right, roller) being able to fire the gun. It is recommended that the player use a separate mouse when playing as it makes the experience better, however trackpad can still be used. The game's development will include creating a mouse-based movement and firing system, as well as modelling the enemies so that they have hitboxes, health points and a score attributed. A GUI will also be made to act as starting screen and a game over screen that shows the total score. Development will be split into creation of game mechanics, development of the enemy Class, implementing GUI, testing the completed game and finally deploying it after 7 days.

Project Schedule:

- **Day 1:** Brainstorming ideas and creating plans for the project
    - Think of game ideas that can be developed in the given time frame and based on current skills/learning ability
    - Start creating mini-outlines for each idea to improve long-term foresight for each idea and making the decision process easier

- **Day 2:** Lay down the non-coding foundation of the project
    - Set up Gantt charts to see how time can and should be allocated
    - Create basic versions of game ideas

- **Day 3-5:** Finalise planning and start training
    - Finalise game decision for the project and create proper outlines
    - Complete Gantt chart based on the chosen game
    - Start PyGame training

- **Day 5-8:** PyGame training and creation of game foundation
    - Learn the basics of PyGame
    - Learn about mouse movement, image movement using a mouse or keys and the use of loops in PyGame
    - Create the IPO and Pseudocode for the game to help with future dev

- **Day 9:** Start the development of the game's player mechanics
    - Write code for the Player Class, implementing the mouse movement aspect of the game
    - Lay the framework for the click to shoot system that will rely on the Player's location for direction
    - Update Gantt Chart and Project Management reports

- **Day 10-11:** Develop the shooting system and integrate Player movement, shooting and background animation into one file
    - Write code for the Bullet Class, which creates bullets at the players location when the mouse is clicked
    - Write code to integrate the Bullet Class with the Player Class, with the bullets now being drawn and “fired” from the Player’s location
    - Write code for the Background Class, which is a series of the same image being repeated to simulate the visual effect of the Player flying over snowy fields (battle is themed after the end of Stalingrad in January)

- **Day 12-14:** Finish development of the Player and create the Enemy Class
    - Test the newly integrated Player, Bullet and Background classes, and make improvements/remove bugs
    - Write code for the generation of enemies in the Enemy Class, which will use similar generation mechanics as the Background Class
    - Write code to define hitboxes for enemies, which will use collision detection
    - Write code to implement a hitpoint system for enemies based on the hitbox collision detection and the Bullet class
    - Write code to tie the hitpoint system directly together with a scoring system that will count points for the player

- **Day 15-17:** Finish the development of the base game, start the development for a GUI and continue developing the scoring system
    - Integrate the new Enemy class with the already made Player, Bullet and Background classes, and test the code to find bugs/make improvements
    - Start the creation of a GUI for the start page, which will have a start button and a help button to show controls, and a GUI for the end page, which will show the player’s final score and ask if they want to leave or replay
    - Develop the scoring system and Enemy class further by writing code to include different enemies with different attributes that influence points

- **Day 18:** Finalise the entire game, run testing and deploy it
    - Test the fully completed game that integrates the GUI and scoring system with the functioning base game that has the Player, Enemy, Bullet and Background class
    - Write code to use images instead of generated shapes to improve “graphics”
    - Fix any bugs and make any minor improvements to improve game mechanics and/or game flow
    - Make final touches to the DFD, Gantt Chart, project outline and IPO/Pseudo code if needed
    - Deploy the game and related files to a public GitHub repository
    - Prepare any documentation needed for users installing the game
    - Prepare for final deployment of the game

Project Budget:

- **Resources and Materials: $0**
    - All resources used for the development and running of the game should require no extra cost, therefore no budget needs to be allocated
- **Labour: $660 ($55*12h)**
    - The labour budget is enough to hire a Python game developer for 12 hours of work, which should be enough time to develop the game code
- **Potential Cost Overruns:**
    - Delays in development: delays in development, especially during the creation of more technical game mechanics, could lead to more budget needing to be spent on labour to account for extra work hours
    - Demand for new features during play-testing: users and playtesters may demand new features be added to the game on top of what is already planned, which may require the purchase of extra resources and materials in order to either commit these changes or simply to support the more complex game

Project Risks:

- **Schedule Risks**
    - Delays in development of the Player, Bullet and Enemy classes
    - Delays in the integration of the aforementioned classes
        - The most likely out of all the schedule risks, as it requires the most time and involves many more variables that could go wrong
    - Delays in integrating the game code with the GUI
    - Delays in debugging
- **Technical Risks**
    - Difficulty in implementing unique hitpoint systems and enemies using PyGame compatible code for the Enemy class
    - Complexity in seamlessly integrating several mechanics together
        - Ties directly into the risk of delays during integration

Project Resources:

- **Human Resources:**
    - Python Programmer well versed in the basics of Pygame and the Python language in general
- **Equipment:**
    - Development Software: This project will use various apps meant to help with software development, such as VS Code to act as the Python interpreter and to debug the code
    - Computer: the project needs a computer to support the software that this code will run on, and also to store the necessary code and image files
- **Extra Resources:**
    - Online Tutorials: in order to undergo worker training, online tutorials such as YouTube videos from professional Pygame users or documentation on Python-related websites will be used to enhance and hasten the learning process in order to stay within time constraints

    - ChatGPT: ChatGPT will be used during the code development process to help explain more technical aspects and to provide a fast way to access resources rather than having to browse the Internet. It will also be used to run simulations of code to see where potential problems could arise

Communication Plan:

- **Key Stakeholders:**
    - The game developer: the game developer will consistently need to be updated with feedback from testing and/or from users who have tried the game in order to improve the game's features
    - The users that playtest: the users will most likely be looking for a functional and smooth yet fun game that doesn't have several game experience ruining bugs. They are crucial to finding and reporting bugs before deployment.
- **Communication Channels**
    - TeamGantt: Project management websites such as TeamGantt, which automatically notifies all team members of upcoming or outlying tasks, can be used to coordinate between project managers and the game development team via email or meetings created by the website. This will also help with time management and task delegation.
- **Communication Milestones:**
    - User feedback updates: frequently receive feedback from users-to-be and playtesters to gain insight on what to fix and what to add
    - Progress reports: frequent progress reports throughout the entire team so that the game developer can be assigned new tasks if needed, and also so the project manager can see how well the project is progressing

Quality Management Plan:

- Develop a bug-free and stable game that won't be too resource intensive
- Provide a fun and engaging experience for the users
- Ensure easy access across multiple platforms, and the ability to easily play
- Have a continuous cycle of user feedback and new improvements to work towards a game that will best satisfy the user demands

Procurement Plan:

- **Procurement Needs:**
    - One Python programmer who is able to develop games using PyGame
    - Acquiring the necessary infrastructure to support development and deployment of the project, mainly up-to-date VS Code, Git and a GitHub repository
- **Procurement Timeline:**
    - Planning: during the planning stage of the project, a suitable programmer will be hired and the necessary infrastructure will be set up. This takes 2-3 days.
    - Training: the programmer will undergo learning to be accustomed with the most up to date aspects of Python, PyGame and GitHub, This takes 1-2 days.
    - Programming: once training and planning is complete, the programming of the core game mechanics, object classes and helper functions will be underway. This takes 5-6 days, with extra time later to fix issues discovered during testing. Creating GUIs to act as start and end screens will take an extra day to develop once base program testing is completed
    - Testing: a version of the base game with all the main mechanics will be tested for around 3-4 days. During this time, game mechanics will be improved, bugs will be removed and inefficiencies will be removed
    - Deployment: The completed code and the images that come with it will be deployed to a public GitHub repository in stages, with development branches being made from the main repo to allow for last minute changes. The programmer will need to use Git in order to commit, push or pull any changes made to the game at this time. All development branches will be merged and closed once the code and necessary documentation are all complete, making it open to the public

Change Management Plan:

There will be lots of time to make changes if necessary, given the lengthy timeframe for both planning and actual development. The plan to make changes resembles this:

1. **Change request is submitted:** stakeholders, playtesters, or developers themselves can submit a request to make a change to the code or the project management
2. **Change request is reviewed:** the project manager will review the change request to see if it feasible with the given resources, budget and timeframe
3. **Verdict for the request:** if any of the parties reject the change, then work will continue on as usual and the requester will be notified. If the change is accepted, it will be made a part of a new project management plan
4. **Implementation and monitoring:** once approved, the implementation of the change will start, and will consistently be monitored by all parties involved in order to make sure that the change is integrated without issues and also so everyone involved knows what is going on
5. **Integration of the change:** necessary documentation for the change will be made. Stakeholders, playtesters and others will be notified of the completed change

Additionally, there will be change control metrics and reporting requirements:

- **Change control metrics**
    - Number of change requests received and processed.
    - Time taken to evaluate and approve/reject change requests.
    - Impact analysis of implemented changes on scope, schedule, and budget.
- **Reporting requirements**
    - Regular change management reports highlighting the status of change requests, including pending, approved, and rejected changes.
    - Communication of the implementation of changes and their impact to all involved stakeholders.

Closure and Evaluation:

- **Closure activities and deliverables**
    - **Delivery of a completed game:** deliver a game that is entertaining with several different features to the customers
    - **Post-delivery testing:** have post-delivery testing to test all game mechanics yet again to receive final feedback from the customers
    - **Project documentation:** include IPO, Pseudocode, the PMP, a DFD, a Gantt chart to outline project planning, structure and timeline. Additionally, submit a readme file for the customers to gain an understanding of the game and how to play it
    - **Final reports:** create final reports on the successes and failures of the project, outlining user feedback, overall completion of goals and any outstanding events, for stakeholders to see
- **Lessons and best practices**
    - **Seamless integration through inheritance:** the use of class inheritance was key for the integration of major classes in the code and reduced the time needed for the integration phase
    - **Frequent use of branches during deployment:** branches were used in the GitHub repository during the deployment stage in order to leave completed products untouched while steadily submitting anything else
    - **Organisation within the filing system:** it was discovered during the initial deployment phase that organising files into dedicated folders would make both project deployment and installation for users easier.
- **Evaluation criteria and metrics**
    - Overall stakeholder and customer satisfaction with the finished product
    - Compliance with scope, requirements and requests
    - Adherence to the set timeframe of the project
    - Overall achievement of goals set for the project