Robosport 370

Requirements Document

Group C3

Corey Hickson - crh208

Levi Paradis - ldp343

Daniel Sanche - drs801

Landon Tetreault - lzt639

Matthew Wind - mdw466

October 7th, 2015

I. Introduction

In this report, we will discuss the CMPT 370 RobotSport370 project. This project will simulate battles between robots, which follows a set of rules, created by players on a virtual map. The project will contain multiple parts that will all have different requirements and restrictions. The battle will then be displayed graphically on a hexagonal map which spectators will be able to watch and possibly manipulate.

The robots will be designed with a specific standard decided in a standards meeting. The project will include designing robots to participate in other teams simulations. The robots will be able to run on any simulation provided it adheres to the standards laid out in the standards meeting.

II. Background

In September, 2015, our group was commissioned by Professor Christopher Dutchyn to design a program, RobotSport370, which would simulate battles between player's robots created on a virtual map. We were given certain requirements, important features, 'could-haves', and extras, which we will outline in our scope.

This project will require a design method to be followed to the successful completion of the project. We have begun this project to meet these requirements and practice the design process.

III. Scope

For this project, the scope contains four different categories of requirements listed below.

A. The software must:
    1. be able to run on the University of Saskatchewan's Linux machines using TuxWorld,
    2. be able to run simulations with robots designed by players. This involves writing an interpreter to interface between the robots and the simulation.
    3. implement a standard set of robot actions, as decided in the robot language specification. These include: "move", "shoot", "scan", "identify", "hex", and possibly others if the standard evolves over time,
    4. implement a graphical user interface that will allow users to view a hexagonal map and see important information about the match,
    5. a set of robots must be created using the the robot language specified in class.

B. The software should:
    1. have the option to run a simulation without displaying graphics to allow many simulations to be run in a short amount of time,
    2. provide useful information about the simulation to the spectators; players could also see information about the amount of time the simulation has been running for or information about the robot's current status and team information,
    3. be cross-platform if possible,
    4. and have a debug/testing simulation in which we can view a match at different speeds, including rewind, fast forward, and viewing stats about each robot.

C. It would be nice if the software could:
    1. let the spectator be able to play and pause the simulation at will,
    2. provide a testing mode with a feature that allows the user to move robots around the map at will so they can experiment with different scenarios,
    3. and do some validation on robots to check for errors. At the basic level, just validating their health/strength values would be sufficient

D. We would like, if time permits:
    1. advanced graphics and sounds, possibly including animations, or detailed sprites for each robot. Complex graphics are considered a low priority, and will only be implemented after the rest of the system is working.
    2. more complex debugging options. Ideally, to be able to step through and edit code as a match plays, so we can experiment with different robot behaviours.
    3. and more complex error handling for robots, including analysis of their program to make sure there won't be errors and reject robots in advance if there are any.

IV. Actors

For this project, we have identified a number of points of interaction with our system, actors. These include:

A. **robots** which must be able interact within our simulation. These robots are external to our system, but we must make sure they are compatible with our interface,
B. **spectators,** external actors that set up and view tournaments,
C. and **testers,** a special case of spectators. They should be able to do everything a spectator can do, but they will also require additional features in the match view, allowing them to experiment with different strategies for the robots.

V. Use Cases

For this project, we have identified a number of use cases. These use cases describe what our actors can do to interact with the system and are listed below. Additionally, a use case diagram follows to visualize the listed use cases.

A. As a spectator, I want to watch a match through a GUI.
B. As a spectator, I want to be able to pause and resume a match as it runs.
C. As a spectator, I want to pick teams of robots for a match.
D. As a spectator, I want to choose the map size.

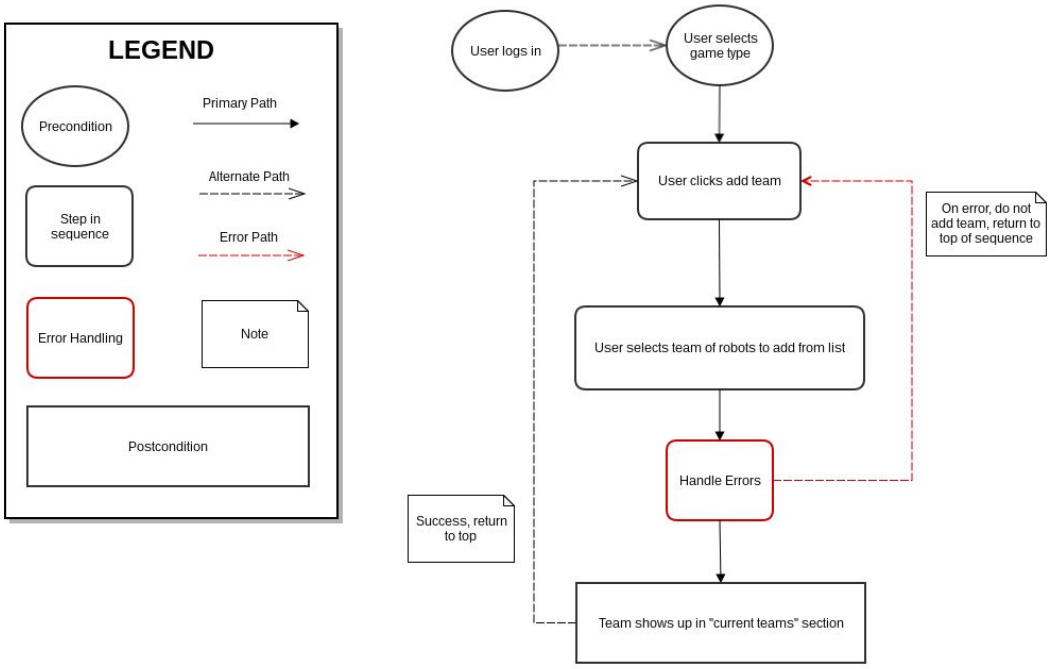
E. As a robot I want to be loaded into the simulator.
F. As a robot I want to be able to shoot.
G. As a robot, I want to be able to move.
H. As a robot, I want to be able to scan for other robots.
I. As a robot, I want to identify information other robots.
J. As a robot, I want to be able to identify details of a given tile.
K. As a robot, I want to be able to send/receive information to my teammate through mailboxes.
L. As a robot, I want to be removed from play when my health reaches 0.

M. As a robot, I want to be able to execute my script on my turn.
N. As a tester, I want to be able to rewind.
O. As a tester, I want to be able to step through instructions.
P. As a tester, I want to be able to edit robot's stats on the fly.
Q. As a tester, I want to be able to reposition robots on the fly.
R. As a tester, I want to simulate many matches simultaneously without a UI.

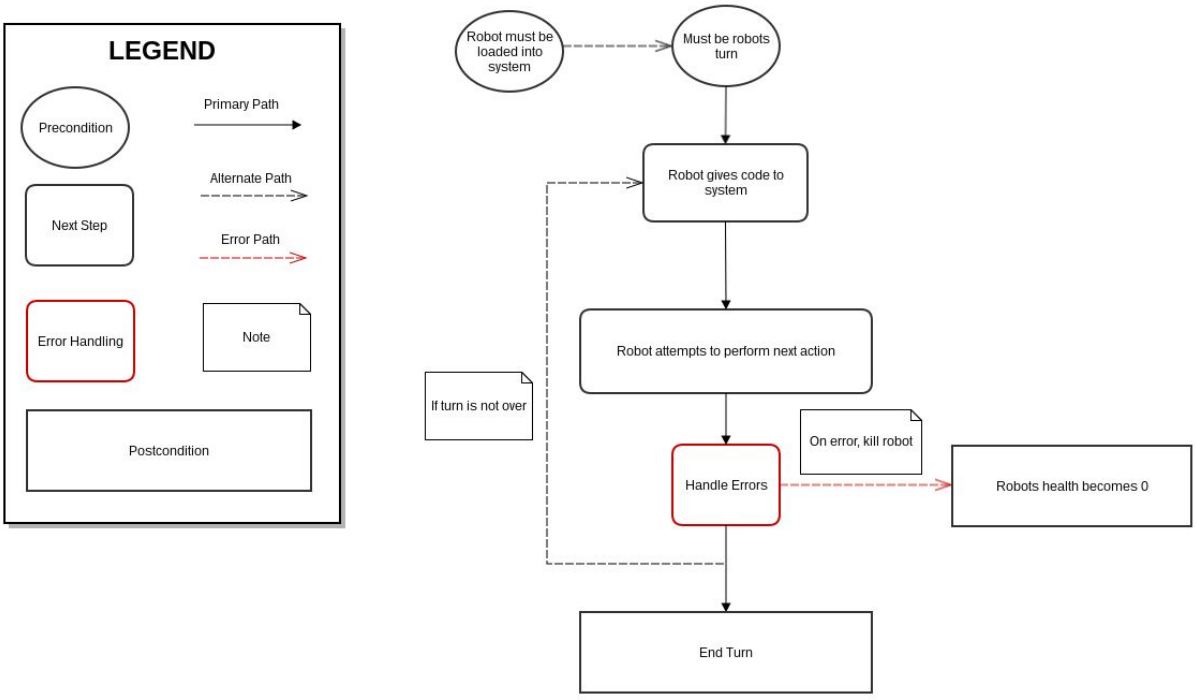
VI. Activity Diagrams

To better understand the project's flow, we have created the following activity diagrams.

A. Load robots/teams into simulator
    1. Preconditions: User must be logged in, User must have selected game type
    2. Flow of events:
        a) User clicks add team
        b) User selects team of robots to add
        c) Handle errors
        d) Robots show up in current robots screen
        e) User can then choose to add more teams (reuse sequence)
    3. Postconditions: Team(s) will be loaded into system ready for simulation
    4. Error Conditions: Team not found
    5. Sequence Diagram



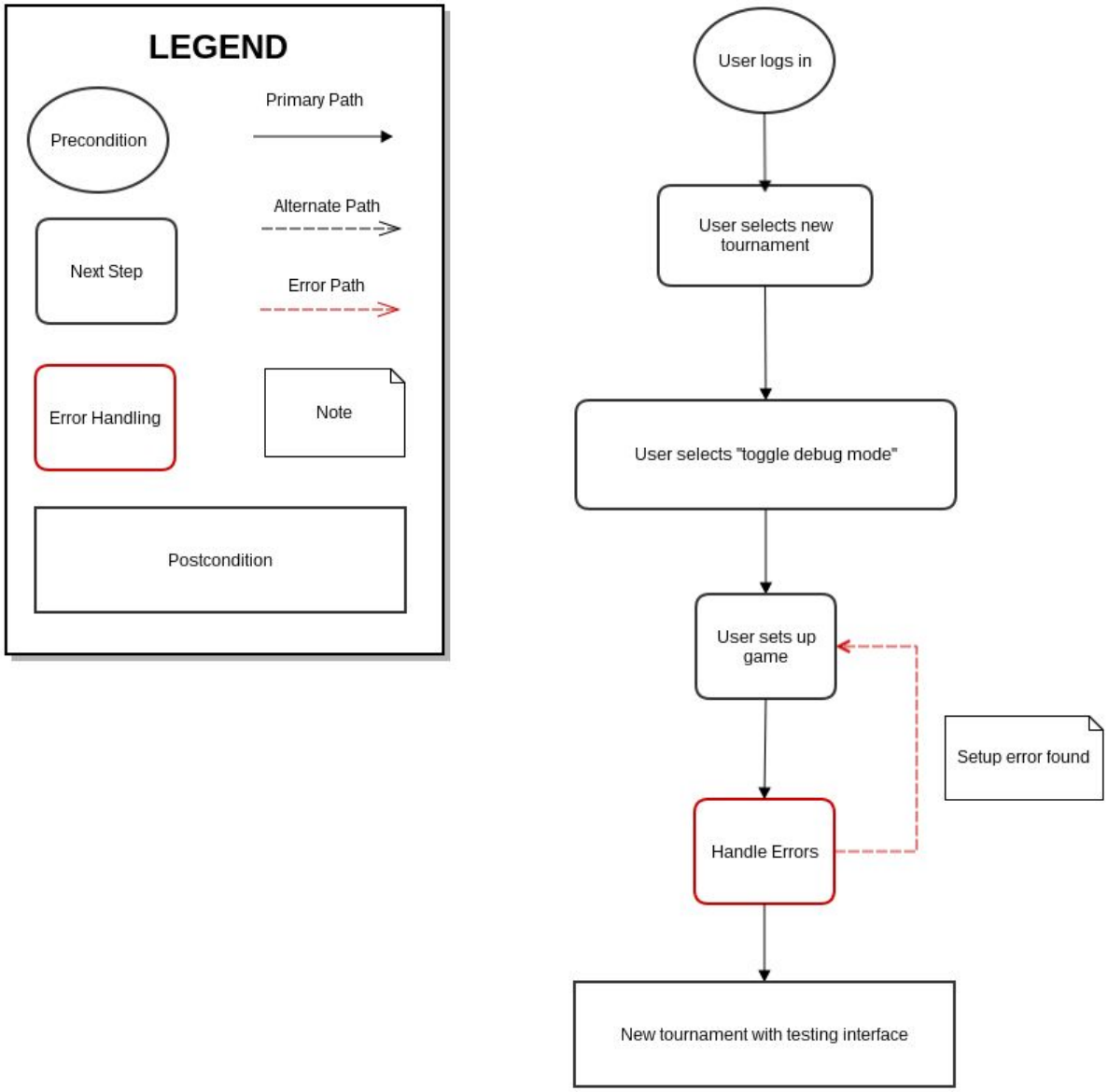
B. Execute Turn
    1. Preconditions: Robot must have been loaded into system, must be in a new tournament, must be robots turn
    2. Flow of events:
        a. Robot gives code to system
        b. Robot tries to perform its next action(s)
        c. Handle errors
        d. If the turn is not over, loop back to top to perform next action
        e. End turn
    3. Postconditions: Robot successfully executes and ends turn
    4. Error Conditions: Robot's program is incomplete, contains errors, tries to cheat
    5. Sequence Diagram:



C. Debug match form GUI
    1. Preconditions: User must be logged in
    2. Flow of events:
        a. User selects new tournament from main menu
        b. User selects error check
        c. User loads robots
        d. User selects map size
        e. Error check
            i. If no errors in setup, start new tournament in debug mode
            ii. If errors in setup, display error message, do not start new tournament

f. User can now debug game using debug interface

3. Postconditions: A new tournament with the testing interface
4. Error conditions: Errors in game setup (loading robots, invalid number of robots/teams)
5. Sequence Diagram:

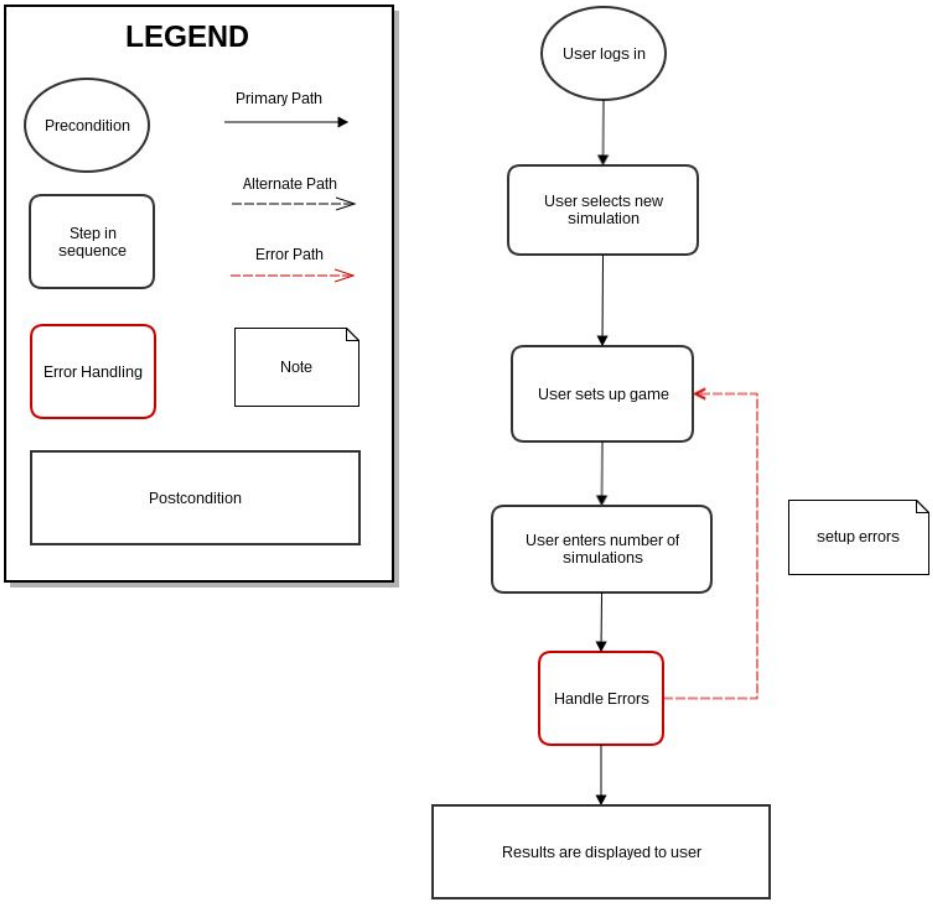


D. Run fast simulation without graphics

1. Preconditions: User must be logged in
2. Flow of events:
    a. User selects new simulation
    b. User sets up game (adds robots, picks map size)

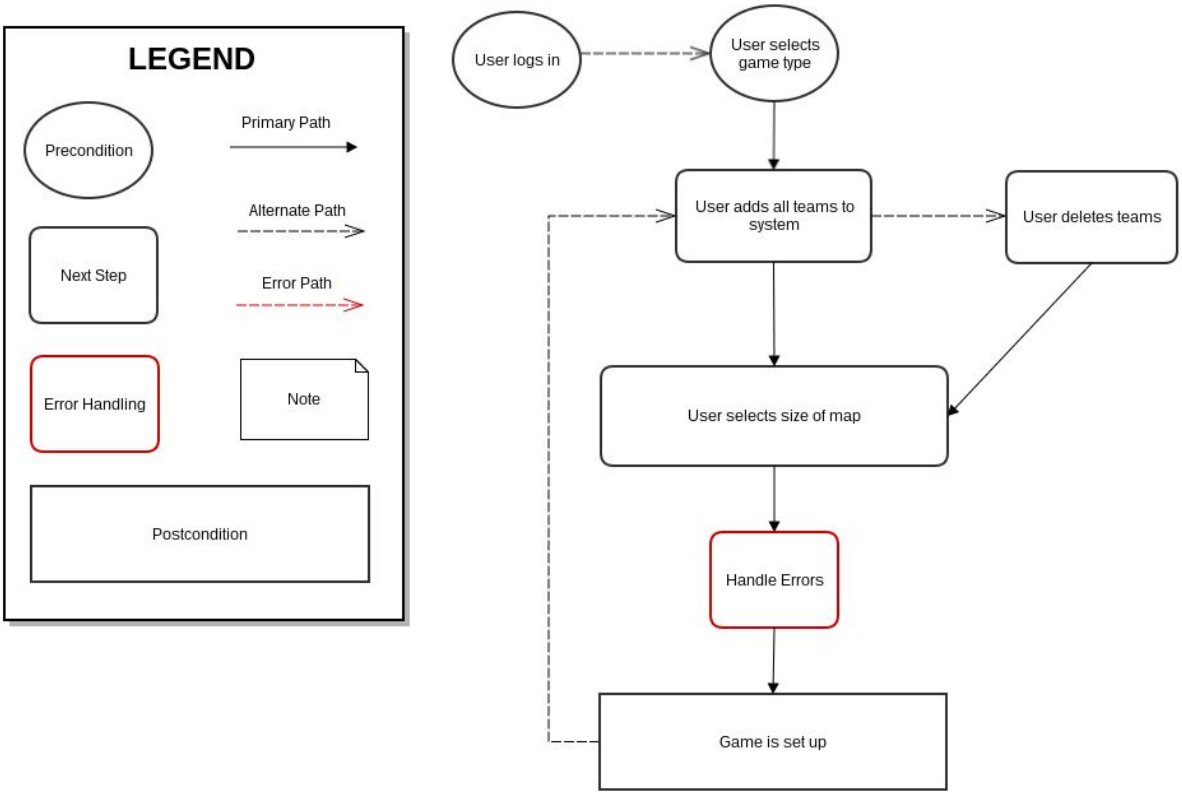
c. User selects how many times they want the simulation to run
d. User views results of all simulations

3. Postconditions: Results of simulation displayed to user
4. Error Conditions: errors in game setup
5. Sequence Diagram:



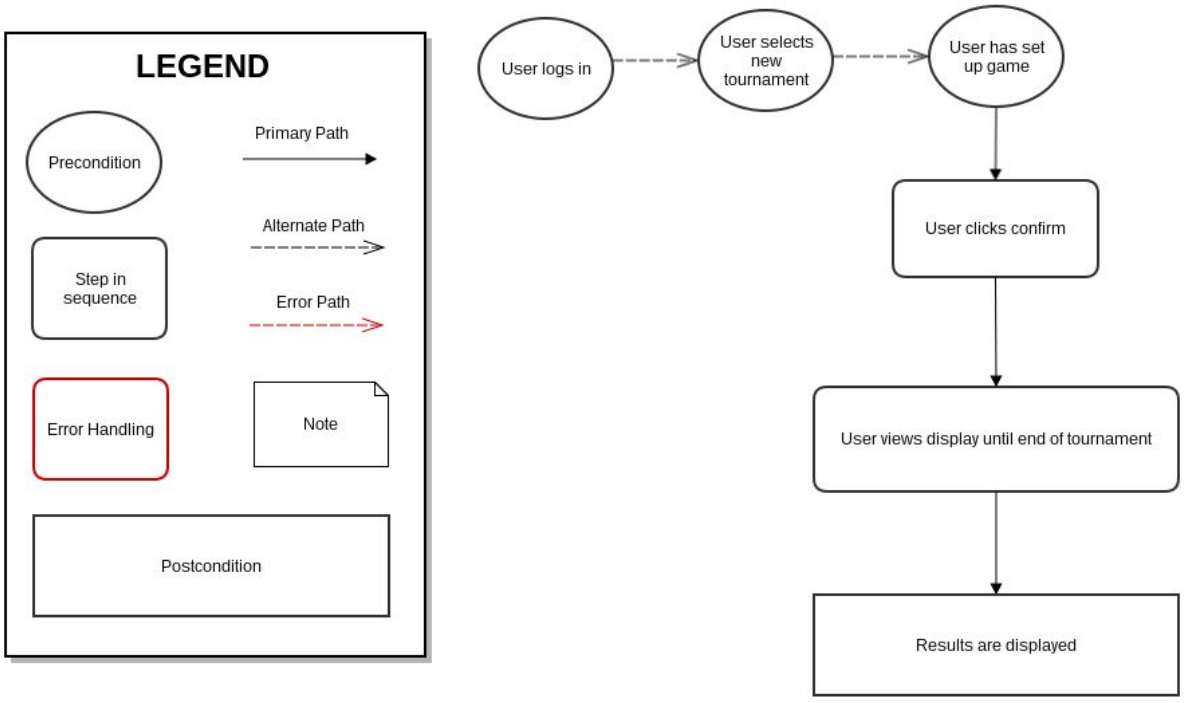
E. Setup match

1. Preconditions: User must be logged in, User must have selected game mode
2. Flow of events:
    a. User adds all teams to system
    b. User selects size of map
    c. Handle Errors
    d. User can optionally delete teams
3. Postconditions: Game is setup to users liking
4. Error Conditions: User trying to select an invalid map size
5. Sequence Diagram:

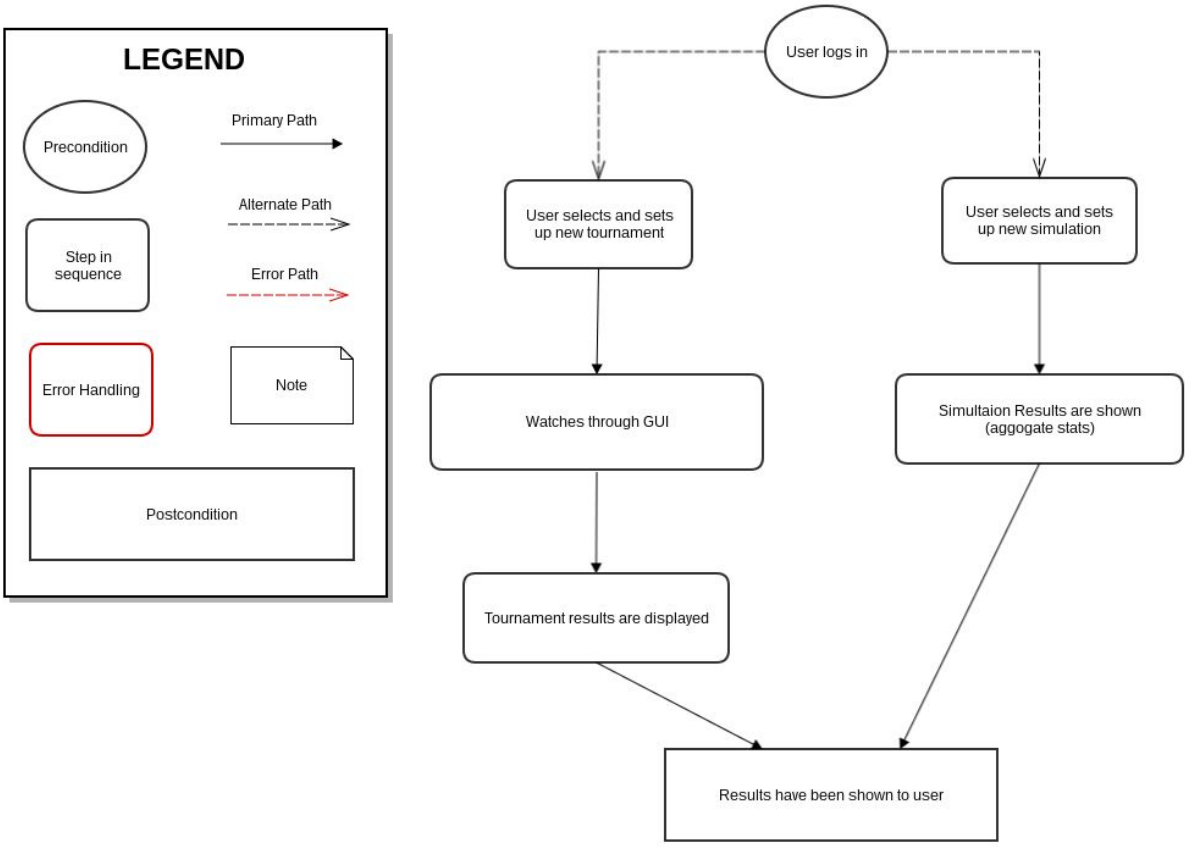


F. Watch match through GUI

1. Preconditions: User must be logged in, user must have selected new tournament, user must have set up game
2. Flow of events:
    a. User clicks confirm and runs game
    b. User views display until end of tournament
3. Postconditions: User will have watched a tournament, display now displays stats from tournament
4. Error Conditions: Errors handled by setup game, User errors (added wrong teams, selected debug mode by accident)
5. Sequence Diagram:



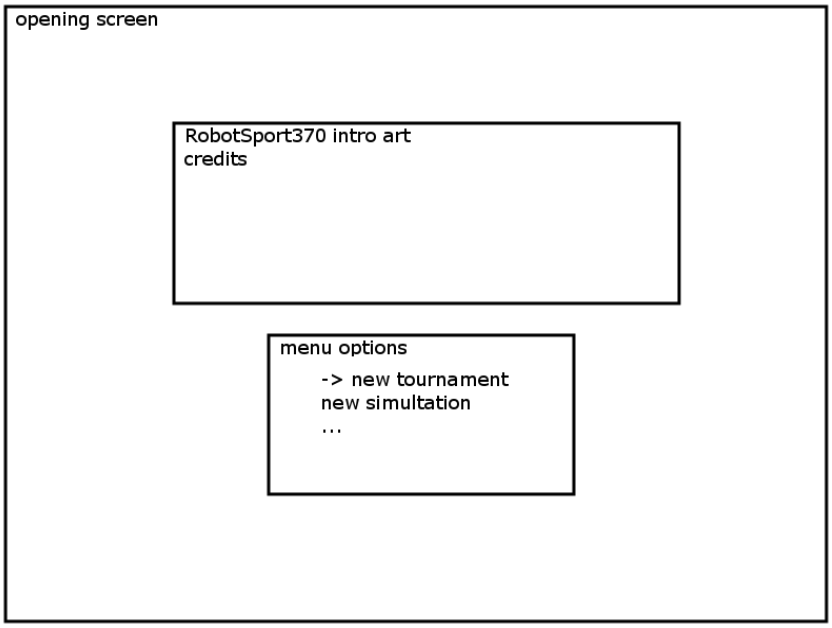
G. View Results of match
   1. Preconditions: User logged in
   2. Flow of events:
      a. User selects new tournament or new simulation
      b. User performs setup
      c. If user selected new tournament
         i. User watches till end of game
         ii. User views results on screen
      d. If user selected new simulation
         i. User gets results right away
         ii. User gets aggregate stats instead of single tournament stats
   3. Postconditions: Display shows results for user to view
   4. Error Conditions: User errors in selecting game mode, setup errors
   5. Sequence Diagram:



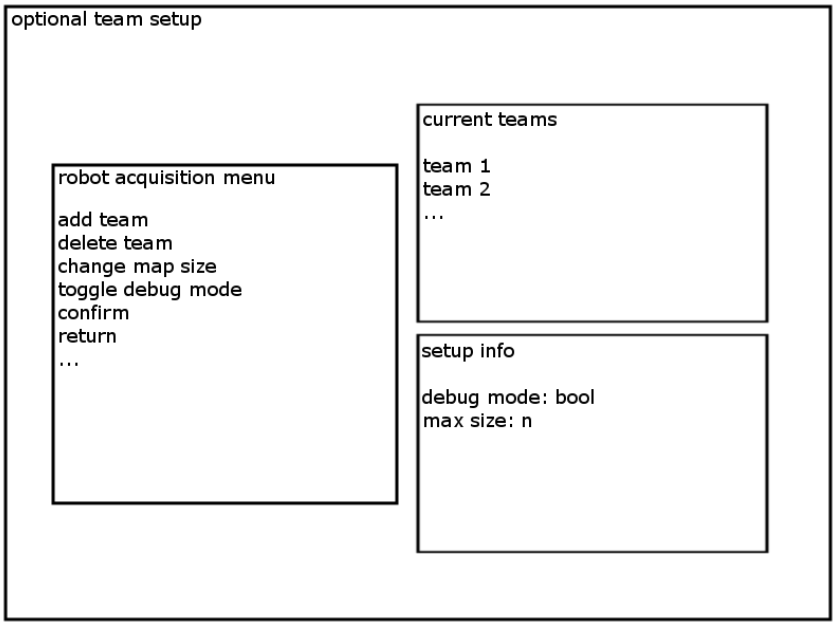
VII. Storyboards

For our storyboards, we came up with five primary screens to create, as below. These storyboards act as a guide for our graphical user interface (GUI).

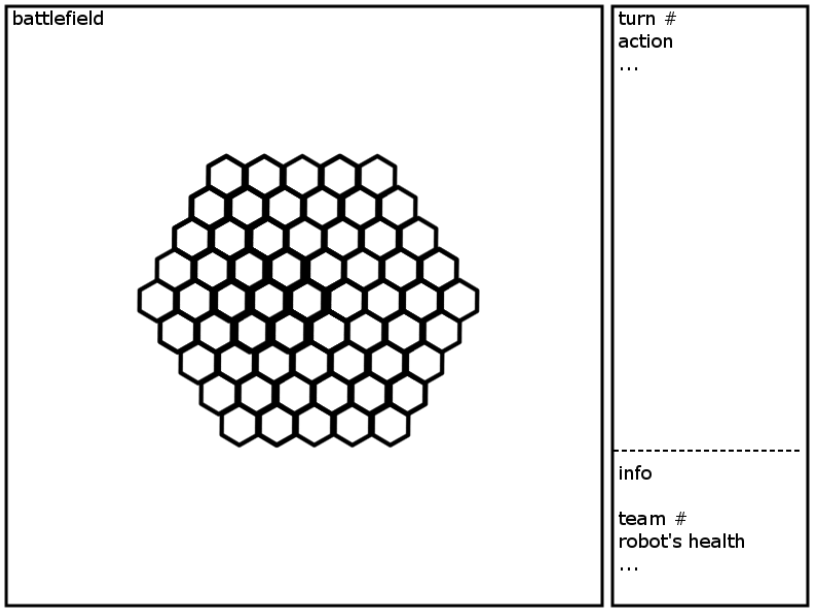
A. The opening screen storyboard is the initial screen of the RobotSport370 game. Upon initialization of the game, the tester or spectator will be give a series of options for starting a specific mode. This will only provide selectable options leading into the following story board such as new game, new tournament and new simulation.



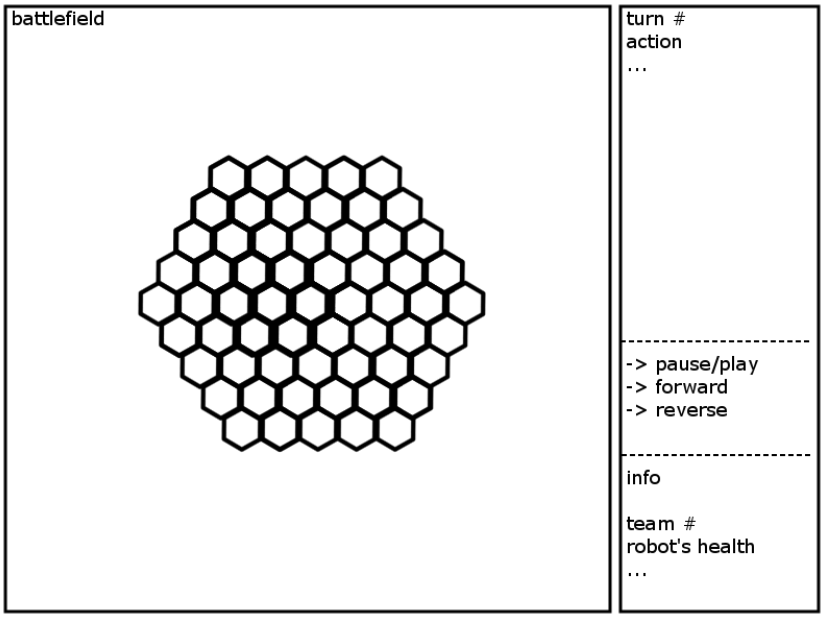
B. The optional robot setup storyboard describes the GUI after the spectator or tester chooses a game mode. This will provide them with the setup for whichever game they are preparing. The setup performed at this screen will include managing of the teams, the map size, and returning or confirming.



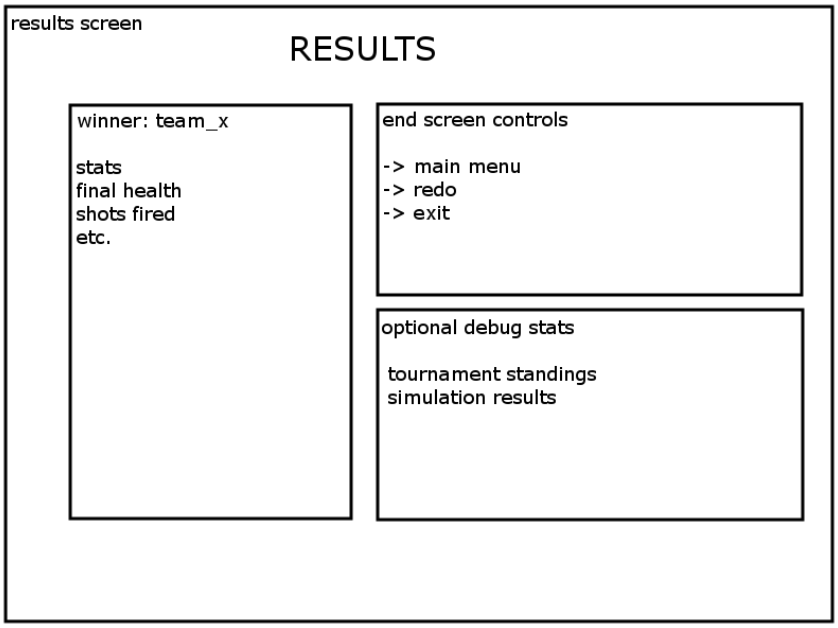
C. This screen will be the typical play screen which spectators will view for regular games or tournaments. It includes a battlefield (zoomed appropriately for the map size), with game information (what is happening), and robot information.



D. The difference for this battlefield storyboard is that it simply includes debugging information. This will be accessible through the debug option on the setup screen.



E. The results screen will appear after a tournament or simulation and will display relevant results and offer controls on continuing the program. It will include relevant stats for the winning team, and potentially other teams. The end screen controls will allow for restarting or exiting.



Additionally, there will be a robot upload GUI, however this will be determined after the standards meeting has happened.

VIII. Conclusion

In conclusion, we will be creating a project which initializes a virtual battlefield and runs simulations. The simulations will be run using robots created by various teams. To do this, we have identified several project requirements.

First, the scope of the project. The scope is split up as follows: must-haves, should-haves, nice-to-have, and would-like-to-have.The project should include at least the must-haves and should-haves.

Next, we defined a list of actors interacting with our program. These actors include robots, spectators, and testers. Our program must be able to interface with all of those actors. Furthermore, this will reduce possible failure points.We have come up with several use cases for our program. They cover possible execution paths the actors could take.

Next, we outlined activity diagrams which go through the possible branches the program could run. Our activity diagrams allow us to fully understand the possible flow of events and help us ensure they are properly executed. We also created storyboards which provide the different mockup graphical user interfaces the program will use.

Finally, this project now outlines the RobotSport370 project and its requirements which we can use to fulfil as we continue the project. Following this document, we can now complete the project.