Robosport 370

Requirements Document

Group C3

Corey Hickson

Levi Paradis

Daniel Sanche

Landon Tetreault

Matthew Wind

October 7th, 2015

I. Introduction

This project will be a simulator that will simulate battles between robots created by other players all on a virtual map with a set of rules that must be followed. The project will contain multiple parts that will all have different requirements and restrictions. The simulator created will be able to simulate battles with other players robots. 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 that will be decided in a standards meeting, this project will include designing a set of 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, called 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 this design process.

III. Scope

A. Must Have:

1. The software must be able to run on the Universities' Linux machines using TuxWorld.
2. The software must be able to simulate matches between other players and be able to run simulations with robots designed by other teams. This involves writing an interpreter to interface between the robot's language, and the simulation's.
3. Standard set of robot actions must be implemented, as decided in the robot language specification. these include "move", "shoot", "scan", "identify", "hex", and possibly others if the standard evolves over time
4. A User Interface will be implemented that will allow users to view the hex map, and see important information about a match. Some sort of graphics will be required, at the very least ASCII graphics will be used, but if time permits more detailed graphics will be made.

5. A set of robots must be created using the the robot language specified in class.

B. Should Have:
   1. The software should be able to have an option to run a simulation without displaying graphics, this would allow many simulations to be run in a short amount of time.
   2. In the user interface having useful information about the simulation to show the spectators would be useful, players could see information such as the amount of time the simulation has been running or information about the robot's current status and team information.
   3. The software should be cross-platform if possible
   4. The program could have a debug/testing simulation in which we can view a match at different speeds, rewind, fast forward, and view stats about each robot.

C. Nice to Have
   1. The spectator should be able to play and pause the simulation at will
   2. The testing mode would benefit from having a feature that allows the user to move robots around the map on the fly, so they can experiment with different scenarios.
   3. The program should do some validation on robots to check for errors. At the basic level, just validating their health/strength values would be sufficient

D. Would Like to Have
   1. If time permits, advanced graphics and sounds would be implemented. These may include 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 would be a desired feature if time is permitting. Ideally, we would like to be able to step through and edit code as a match plays, so we can experiment with different robot behaviours.
   3. More complex error handling for robots could be implemented, where we do analysis of their program to make sure there won't be errors, and reject robots in advance if there are any.

IV. Actors

A. External Robots

robots must be able to run in our simulation. These robots are external to our system, but we must make sure we are compatible with the standard robot interface.

B. Spectators

Spectators are external actors that set up and view tournaments.

C. Testers

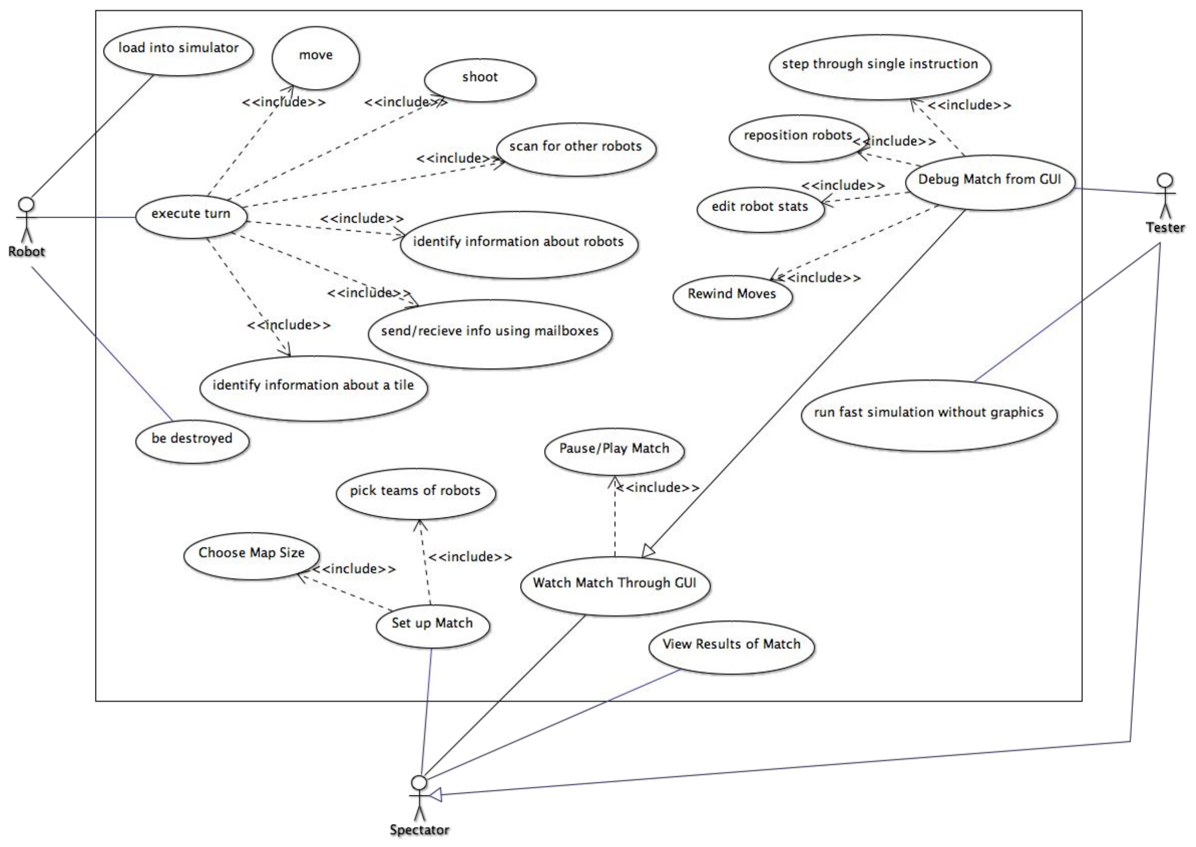
Testers are a special case of spectators. They should be able to do everything a spectator can do, but they will also require additional features in the match view, allowing them to experiment with different strategies for the robots.

V. Use Cases

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/recieve 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



Use Case Diagram

VI. Activity Diagrams

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

B. Execute turn
   a. Preconditions: Robot must have been loaded into system, must be in a new game, must be robots turn
   b. Flow of events:
      i. Robot gives code to system
      ii. Robot tries to perform its next action(s)

            iii. Handle errors
            iv. If turn not over, Loop back to top to perform next action
            v. End turn
    c. Postconditions: Robot successfully executes and ends turn
    d. Error Conditions: Robots program is incomplete, contains errors, tries to cheat

C. Debug match from GUI
    a. Preconditions: User must be logged in
    b. Flow of events:
        i. User selects new game from main menu
        ii. User selects debug mode
        iii. User loads robots
        iv. User selects map size
        v. Error check
            1. If no errors in setup, start new game in debug mode
            2. If errors in setup, display error message, do not start new game
        vi. User can now debug game using debug interface
    c. Postconditions: A new game with the testing interface
    d. Error conditions: Errors in game setup (loading robots, invalid number of robots/teams)

D. Run fast simulation without graphics
    a. Preconditions: User must be logged in
    b. Flow of events:
        i. User selects new simulation (tournament?)
        ii. User sets up game (adds robots, picks map size)
        iii. User selects how many times they want the simulation to run
        iv. User views results of all simulations
    c. Postconditions: Results of simulation displayed to user
    d. Error Conditions: errors in game setup

E. Setup match
    a. Preconditions: User must be logged in, User must have selected game mode
    b. Flow of events:
        i. User adds all teams to system
        ii. User selects size of map
        iii. Handle Errors
        iv. User can optionally delete teams
    c. Postconditions: Game is setup to users liking
    d. Error Conditions:

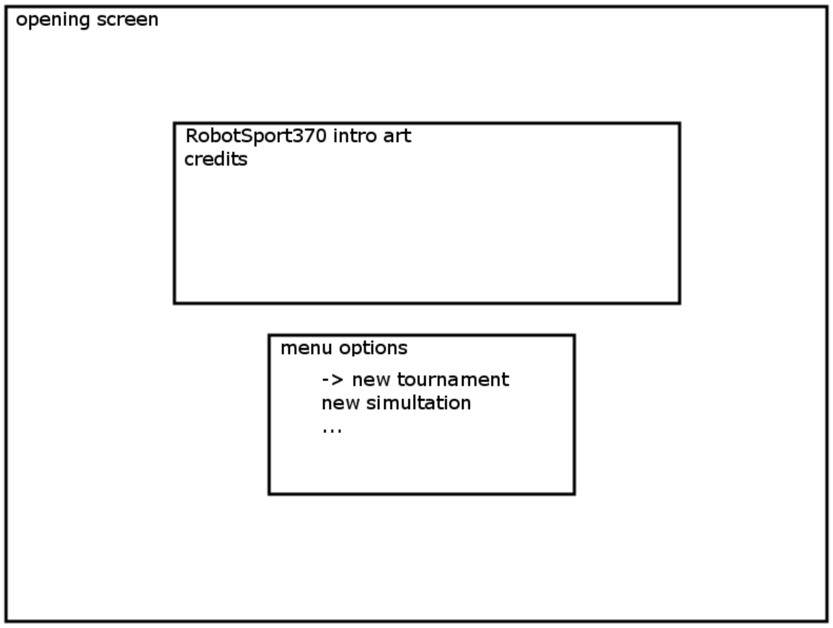
F. Watch match through GUI
    a. Preconditions:
    b. Flow of events
    c. Postconditions:
    d. Error Conditions:

G. View Results of match

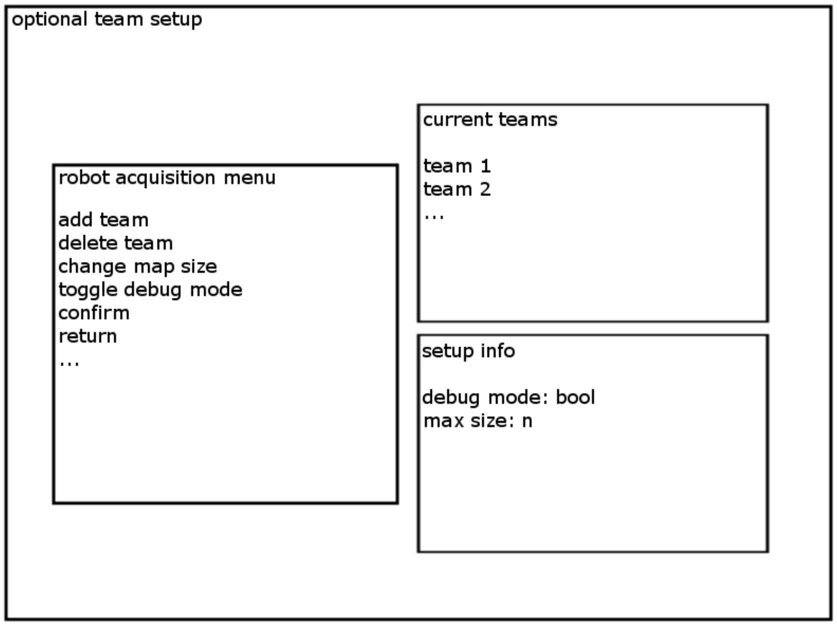
a. Preconditions:
b. Flow of events
c. Postconditions:
d. Error Conditions:

VII. Storyboards

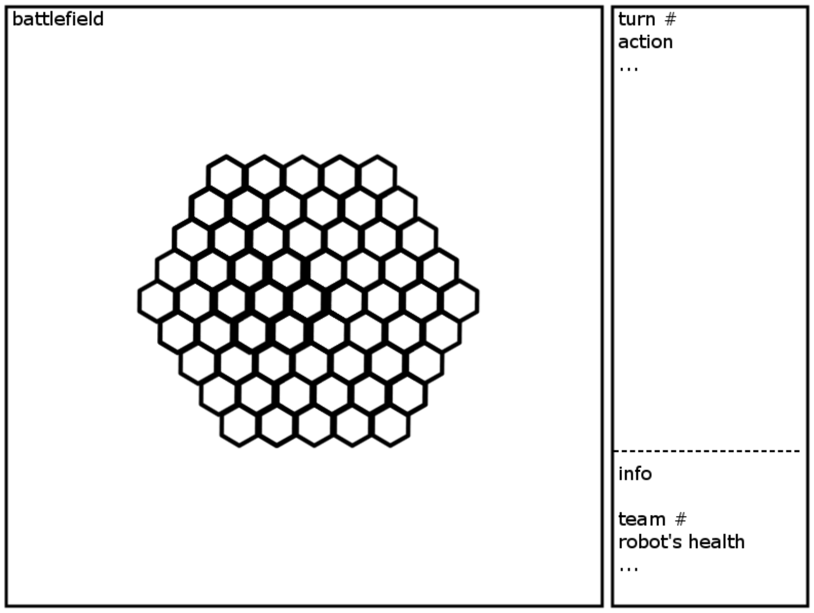
For our storyboards, we came up with five primary screens to create, as below. These storyboards act as a guide for our graphical user interface (GUI).



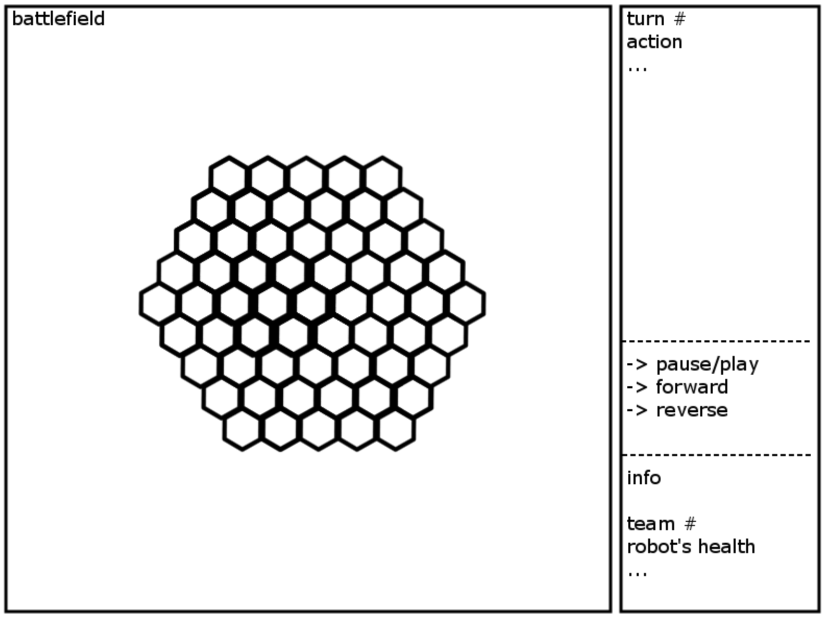
The opening screen storyboard is the initial screen of the RobotSport370 game. Upon initialization of the game, the tester or spectator will be give a series of options for starting the game for a specific mode. This will only provide selectable options leading into the following story board such as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



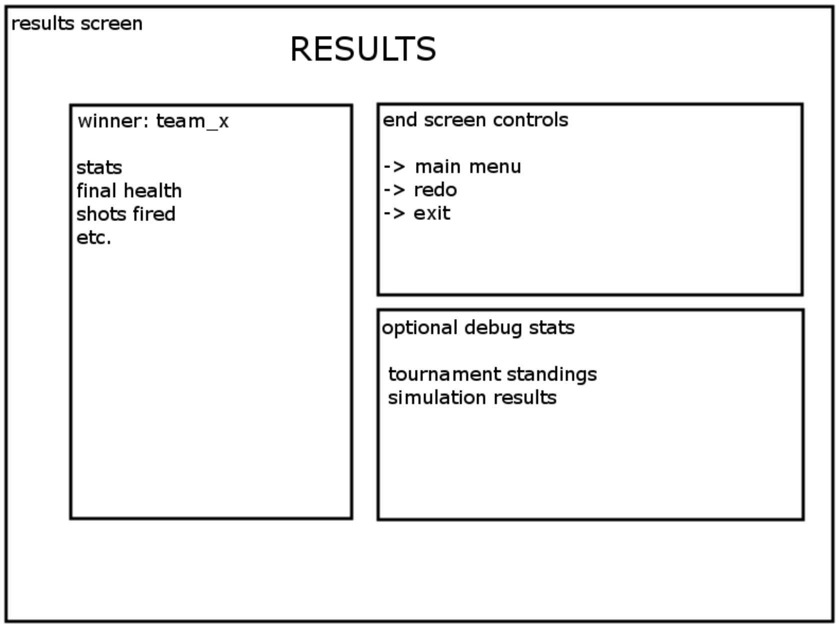
The optional robot setup storyboard describes the GUI after the spectator or tester chooses a game mode. This will provide them with the setup for whichever game they are preparing. The setup performed at this screen will include managing of the robots, map size, and returning or confirming.



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.



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,



The results screen will appear after a game, simulation or tournament 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

VIII. Conclusion

In conclusion we will be creating a project that creates a virtual battlefield and then runs simulations on it. The simulations will be run with robots that will have been created by ourselves and other teams. In order for this project to be a success there are several things that we have done, and the first is that we have set a scope of things that we will include in our program. The scope is split up as follows; must haves, should haves, nice to have, and would like to have. In order to be successful we should be able to complete at least up to the end of the should haves. To help us complete the project we have also defined a list of actors that will be interacting with our program, these include external robots, spectators, and testers. We must ensure that all of the actors can properly interface with our program for it to be usable and not have significant failure points. Also we have come up with several use cases for our program that cover all of the possible actions that might need to be taken while it is running. Following the use cases we have outlined activity diagrams that go through the flow of possible branches that the program could run. Our activity diagrams allow us to fully understand the possible flow of events and ensure that they are executed properly. To be thorough we have also given storyboards that provide the different use interfaces that will be presented while the program is being executed. Following the layout that we have provided we should be able to successfully complete this product by the set completion date.