Requirements Definition

1. Introduction and Context

This document describes the functional and non-functional requirements for the *Brilliant Students vs. Zombie Professors* distributed application that the Spring 2014 CS5200 class is going to build. This game, nicknamed here *BSvPZ*, is a contest between two classes of autonomous agents (students and zombies), several different kinds of supporting agents, in a closed network environment where the students agents have to work together to accomplish a common task -- destroy all the zombies. To do this, they need to create and share resources using other limited, temporal other resources, e.g. time.

The storyline for the game is as follows. The campus has been taken over by mutant zombie professors who are intent on eating the brains of brilliant students. The only hope is to destroy them with bombs made out of "lame excuses" and "whining twine". Since brilliant students come by neither of these things naturally, they have to obtain them from "excuse generators" and "whining spinners." Different kinds of bombs can be made by different quantities of excuses and whining. For example, bombs with made from lots of excuses and a just a little whining have a big impact on zombies but only a short range, i.e., a student cannot toss them very far. A bomb made from just one excuse and lots of whining has less of impact, but a greater range. Gathering excuses and whining, as well building the bombs, takes time and coordination. See the *Conceptual Overview* for more a detail description of the game.

2. Users and their Goals

There are different three actors (kinds of users): game promoter, player, and referee. See Figure 1. The game promoter is a user who will create a new game, either with a default set of parameters or with a specific, custom set of parameters. The game promoter will also announce a game to potential players and allow them to join by registering their automated agents. A player will participate in a game by registering one or more automated agents, including at least one *Brilliant Student* agent. The *Referee* will be the one to start and stop the game, and ensure that any errant agents are removed from the game.

3. Functional Requirements

1 The Game. A game is an instance of the system being run.

1.1 A *Promoter* (a user) must be able to create a new game and give it a label or name.

1.1.1 After a promoter creates a new game, it should register itself with a *Game Registry*, using the game's label and a publically accessible end point. See Req. Def. 2

1.2 A *Game* needs to keep track of settings (configuration parameters) for all the other basic components. Appendix A lists some of the possible configuration parameters.

1.3 In the process of creating or setting up a new game, a *Promoter* must set up configuration parameters.

1.3.1 The Promoter user be able to use a default configuration parameters

1.3.2 The Promoter should be save / load a set of the configuration parameters

1.3.3 The Promoter should be able to view/edit the configuration parameters

1.3.4 Any agent should be able to ask the game for the current game parameters at any time.

1.4 A *Real Player* should be able to run as many *Brilliant Student*, *Excuse Generator*, and *Whining Spinners* agents as she/he would like. Each of these agents should in turn discover what games are *Available* and either allow the *Real Player* which game to join or choice one automatically.

1.4.1 A *Real Player's* agents must be identified by real-world identifiers, e.g., Student A#, First Name, and Last Name.

1.4.2 To join a game, an agent must initiate "join game" conversation with a game, via its public endpoint.

1.5 Once the minimum requirements for the game are met, the *Game* can automatically start itself.

1.5.1 When a Game starts, it will initiate a "start game" conversation with all participating agents.

1.6 A user who is not the *Promoter* or a *Real Players* should be able to register as *Referee* for the game.

1.7 After the game starts, the *Referee* can monitor the agents and remove any agent that appears to be misbehaving.

1.8 A game will automatically determine when it is over and who the winner its.

1.8.1 The winner of a game is the *Brilliant Student* that is still alive and that has the most hits.

2 Game Registry.

2.1 There should a stand-alone (and web-based) component that acts as a game registry which keeps track of all existing games, their publically-accessible end points and status.

2.2 After a game is created, it should register itself with the *Game Registry*, using the game's label, a publically accessible end point, and its status. The initial status should "Avaiable".

2.3 After a game has started, it should information the *Game Registry* that its status has changed to "Started"

2.4 After a game has ended, it should information the Game Registry that it status has changed to "Completed" and who the winner was.

2.5 Any agent can ask the *Game Registry* for a list of *Available*, *Started*, or *Completed* games. The list should include the game's endpoint if it is *Available* or *Started* and the winner if it is *Completed*.

2.6 The Game Registry may purge information about old *Completed* games.

3 Clock Tower and Time Ticks.

3.1 Certain actions or tasks in the game, such as moving, will require time-based tokens, called Time Ticks. In some ways, these tokens are like currency. Agents will must have them to complete tasks what "cost" something.

3.2 A *Clock Tower* will send *Time Ticks* to the agents on a regular basis, e.g. one every 200 milliseconds. The actual frequency is a configuration parameter.

3.3 A valid *Time Tick* must be protected again tampering, replication, or re-use, so each *Time Tick* will include a logical timestamp and a hash code.

3.4 An agent may only use a *Time Tick* once, and may not duplicate or delegate it to another agent.

3.5 The *Clock Tower* will be able to valid Time Ticks and the at-most-once usage by any agent.

3.5.1 When an agent performs some action with *Game* or *Playfield*, then the *Game* or *PlayingField* will validate the associated *Time Ticks* with the *Clock Tower*

4 **Basic Components.** The system needs to support the following components as independent, autonomous agents (things that act on their own to achieve certain goals).

4.1 ***Playing Field*.** Every game will take place on a 3D *Playing Field*. The lowest plane of the *Playing Field* represent will the ground and will be divided into squares of equal size. Some of the squares will represent sidewalk and others will represent grass. *Brilliant Students* and *Zombie Professors* can be in any square, but *Excuse Generators* and *Whining Spinners* can only be on grassy squares. The higher planes of the *Playing Field* represent air.

4.1.1 A *Playing Field* must be able to accept an *Excuse Generator* to be placed in an unoccupied grassy square prior to the start of a game.

4.1.2 A *Playing Field* must be able to accept a *Whining Spinner* to be placed in an unoccupied grassy square prior to the start of a game.

4.1.3 A *Playing Field* must be able to allow a *Brilliant Student* to be placed in any square prior to the start of a game.

4.1.4 A *Playing Field* must be able to accept a new *Zombie Professor* stepping into any square on edge of the playing field any time after a game has started

4.1.5 A *Playing Field* must be able to respond valid *Move* requests from *Brilliant Students* and *Zombie Professors*

4.1.5.1 A valid *Move* request is one that contains a valid *Time Tick* (see Req. Def. 3) that is not more than *x* ticks old, where *x* is a configuration parameter

4.1.6 A *Playing Field* must be able to receive valid bombs thrown from *Brilliant Students* and hit *Zombies Professors* with those bombs, if they are on target.

4.1.6.1 A valid bomb is one construct as follows:

4.1.6.1.1 The bomb contains one or more valid *Excuses*. See ???

4.1.6.1.2 The bomb contains one or more valid piece of *Whining Twine*. See ??

4.1.6.1.3 The bomb was put together with a valid *Time Tick* that is not more than *x* ticks old, where *x* is a configuration parameter.

4.1.6.1.4 All the *Time Ticks* for the Excuses and *Whining Twine* are less or the bomb's *Time Tick*, but not more than *y* ticks older than bomb's *Time Tick*, where *y* is configuration parameter

4.2 **Brilliant Students.** A *Brilliant Student* agent acts on its own to a) move around the playing field, b) gather excuses, c) gather whining twine, d) build bombs, and e) destroy zombies.

4.2.1 A Brilliant Student starts off with some initial *Strength Points, Sidewalk Speed Factor*, *and Grass Speed Factor*.

4.2.2 A *Brilliant Students* can ask the *Game* for all the game's configuration parameters.

4.2.3 A *Brilliant Student* can ask the *Game* for the game's *Playing Field*.

4.2.4 A *Brilliant Student* can ask the *Game* for its *Playing Field's* layout, i.e., which squares are grass and which are sidewalk.

4.2.5 A *Brilliant Student* can ask the *Game* for a list of *Zombie Professors* currently on the field. Information about the each Zombie should include its current location, *Strength Points*, and S*peed*.

4.2.6 A *Brilliant Student* can ask the *Game* for a list of *Zombie Professors* currently on the field. Information about the each zombie should include its identifier, communication end point, current location, *Strength Points*, and *Speed*.

4.2.7 A *Brilliant Student* can ask the *Game* for a list of *Brilliant Students* currently on the field. Information about the each should include its identifier, communication end point, current location and *Strength Points*.

4.2.8 A *Brilliant Student* can ask the *Game* for a list of *Excuse Generators* currently on the field. Information about the each should include its identifier, communication end point, location and *Strength Points*.

4.2.9 A *Brilliant Student* can ask the *Game* for a list of *Whining Spinner* currently on the field. Information about the each should include its identifier, communication end point, location and *Strength Points*.

4.2.10 A *Brilliant Students* gathers excuses, one at a time, by initiating a conversation with an *Excuse Generator* and requesting an excuse. During the conversion, the *Brilliant Student* must provide the *Excuse Generator* with a valid *Time Tick* to receive an excuse. The *Time Tick* has to be one that was not use for any other purpose.

4.2.11 A *Brilliant Students* gathers excuses, one at a time, by initiating a conversation with a *Whining Spinner* and requesting an excuse. During the conversion, the *Brilliant Student* must provide

the *Whining Spinner* with a valid *Time Tick* to receive an excuse. The *Time Tick* has to be one that was not use for any other purpose.

4.2.12 A *Brilliant Students* can use a *Time Tick* to build a bomb from excuses and twine that it already holds. The *Time Tick* has to be one that was not use for any other purpose.

4.2.13 A *Brilliant Student* can communicate the *Playing Field* to throw a bomb in a certain direction and distance. This requires some a *Time Tick*. The *Time Tick* has to be one that was not use for any other purpose. That maximum distance that the bomb will travel is restricted by the amount of twine in the bomb.

4.2.14 A *Brilliant Student* can communicate with the *Game* to move squares. This requires some *Time Ticks*. How fact the student moves is a function of its current *Strength Points*, the type of square it's currently on, and either the sidewalk speed factor or the grass speed factor.

4.2.15 A *Brilliant Student* can communicate with another *Brilliant Student* to exchange information about who what zombies are throwing bombs at.

4.2.16 If a *Zombie Professor* is on the same square as a *Brilliant Student*, it will eat the *Strength Points* of that of that *Brilliant Student* at some rate, via the *Game*.

4.2.17 If the *Strength Points* of a *Brilliant Student* reaches zero, then it is destroyed and the *Game* will create a *Zombie Professor* on that square some amount of time later. The delay is control by game configuration parameter.

4.2.18 A *Brilliant Student* can use *Time Ticks* that haven't been used for any other purpose to heal (i.e., increase its current *Strength Points*, but not beyond its initial amount. The number of *Time Ticks* it takes for one *Strength Point* is defined by a game configuration parameter.

4.2.19 A *Brilliant Student* must be able to respond to requests from the *Referee* or *Monitor* for information about its status. This includes its location, strength, the number of bombs that it is holding, the number excuses (not yet not bombs), the number of pieces of twine that it is holding (not yet in bombs), and the number of used *TimeTicks* that it is holding.

4.3 **Excuse Generators.** An *Excuse Generator* listens for time ticks from the Clock Tower and on each tick makes progress towards building an excuse. It also keeps a queue of completed excuses and responds to request for excuses from *Brilliant Students* on a first-coming, first-serve basis.

4.3.1 A *Excuse Generator* will start off will a specific number of *Strength Points* and *Excuse Creation Rate*. See game parameters in Appendix A.

4.3.2 Over time, an *Excuse Generator's Excuse Creation Rate* will increase according to the *Excuse Creation Acceleration*. See game parameters in Appendix A.

4.3.3 An *Excuse Generator* can ask the *Game* for all the game's configuration parameters.

4.3.4 If a *Zombie Professor* is on the same square as an *Excuse Generator*, it will eat the *Strength Points* of that of that *Excuse Generator* at some rate, via *Game*.

4.3.5 If the *Strength Points* of an *Excuse Generator* reaches zero, then it is destroyed.

4.3.6 An *Excuse Generator* does not have to initiate conversations with any other components, but will listen for and participate in conversations.

4.3.7 An *Excuse Generator* must response to requests from *Brilliant Students*, either by eventually granting valid requests, telling the *Brilliant Students* there are no available excuses with allotted time (before the *Time Tick* becomes invalid), timing out when the *Time Tick* become invalid, or reporting that the request is invalid.

4.3.8 An *Excuse Generator* must be able to respond to requests from the *Referee* or *Monitor* for information about its status. This includes its location, strength, and the number excuses that it is holding.

4.4 **Whining Spinners.** A *Whining Spinner* listens for time ticks from the Clock Tower and one each tick makes progress towards building some whining twine. It also keeps a queue of completed whining twine and responds to requests for whining twine from *Brilliant Students* on a first-coming, first-serve basis.

4.4.1 A *Whining Spinner* will start off will a specific number of *Strength Points* and *Twine Creation Rate*. See game parameters in Appendix A.

4.4.2 A *Whining Spinner* can ask the *Game* for all the game's configuration parameters.

4.4.3 Over time, a *Whining Spinner's Twine Creation Rate* will increase according to the *Twine Creation Acceleration*. See game parameters in Appendix A.

4.4.4 If a *Zombie Professor* is on the same square as a *Whining Spinner*, it will eat the *Strength Points* of that of that *Whining Spinner* at some rate, via the *Game*.

4.4.5 If the *Strength Points* of a *Whining Spinner* reaches zero, then it is destroyed.

4.4.6 A *Whining Spinner* does not have to initiate conversations with any other components, but will listen for and participate in conversations.

4.4.7 A *Whining Spinner* must response to requests from *Brilliant Students*, either by eventually granting valid requests, telling the *Brilliant Students* there are no available excuses with allotted time (before the *Time Tick* becomes invalid), timing out when the *Time Tick* become invalid, or reporting that the request is invalid.

4.4.8 A *Whining Spinner* must be able to respond to requests from the *Referee* or *Monitor* for information about its status. This includes its location, strength, and the number of pieces of that it is holding.

4.5 **Monitor.** Each game will include a monitor that displays the game's state.

4.5.1 The *Game* must keep the *Monitor* update on the current status of the agents, zombies, bombs in flight, hits, and other statistics.

4.5.2 The monitor should display the hits for each *Brilliant Student*, i.e., the number of strength points of damage each *Brilliant Student* inflicts on *Zombie Professors*.

4.5.3 The monitor needs to display end-of-game statistics to the game.

4.6 **Zombie Professors**. A *Zombie Professor* is an automatous agent that will enter the playing field from the one of the edges and seek to destroy *Brilliant Students*, *Excuse Generators*, and *Whining Spinners*.

4.6.1 The game will create new *Zombie Professors* at some creation rate. That rate will accelerate over time. See the game configuration parameters in Appendix A.

4.6.2 A *Zombie Professor* will start off with a specific, but randomly selected number of *Strength Points* and *Speed Factor*.

4.6.3 A *Zombie Professor's* actual speed will be a function of its current *Strength Points* and the type of squares its moving on

4.6.3.1 A *Zombie Professor* will be able move faster (or at least differently) on sidewalks than on other places. See game configuration parameters.

4.6.4 When a *bomb hits a Zombie Professor*, its *Strength Points* will decrease based number excused in the bomb and the amount of damage each excuse is configured to have. See game configuration parameters.

4.6.5 When a *Zombie Professor* is on the same square a *Brilliant Student*, *Excuse Generator*, or *Whining Spinner*, it will "eat" away at that agents *Strength Points* at the zombie's *Eating Rate*, which is randomly chosen between some minimum and maximum range. See configuration parameter.

4.6.5.1 It will eat away at the agent(s) on the same square by communicating with the *Game*, which in turn will communicate with the agent(s).

4.6.5.2 If the *Strength Points* of an agent that a *Zombie Professor* is eating reaches zero, then that agent is destroyed and the Strength Points of the *Zombie Professor* is increased by a factor that corresponds to the type of agent. See configuration parameters in Appendix A.

4. Non-functional Requirements

1. The system will be built in following stages: a) communication protocol design and prototyping, b) middleware, c) resource management and game coordination, d) optimization, and e) testing and refinement.
2. The system's communication protocols must be accurately described in a concise, understandable, up-to-date document.
3. The system's must support agents written on different platforms

5. Future Features

Below is a list of ideas for future features:

- Allow agents to find “power-ups” or things in the playing that increase their strength, speed, or other capabilities.

6. Glossary

This section contains a list important terms and their definition

Appendix A – Configuration Parameters

- Playing field size (width and length in squares). The number of planes (height) is virtual need not be specified.
- Minimum number *Brilliant Student* agents that have to be registered before the game can be started.
- The minimum number *Excuse Generators* agents that have to be registered before the game can be started
- The minimum number *Whining Spinners* agents that have to be registered before the game can be started
- The maximum number of *Brilliant Student* allowed to be registered
- The maximum number of *Excuse Generators* agents allowed to be registered
- The maximum number of *Whining* agents allowed to be registered
- The amount of damage potential (in *Strength Points*) each excuse in a bomb has on a *Zombie Professor*
- The throwing distance (in terms on squares) each piece whining twine adds to a bomb
- The splash diffusion factor that indicators what percentage much of a bomb's remaining strength (excuse damage potential) can be spread out to adjacent squares if the zombies on the current squares have all been killed.
- *Zombie Professor* creation rate
- *Zombie Professor* creation acceleration
- *Zombie Professor* strength range (min to max)
- *Zombie Professor* speed range (min to max)
- *Zombie Professor* sidewalk speed
- *Zombie Professor* grass speed factor
- *Brilliant Student* sidewalk speed factor
- *Brilliant Student* grass speed
- *Zombie Eating Rate Ranging* (min to max), in terms of *Strength Points* per *Time Tick*
- Initial *Brilliant Student* strength points
- *Brilliant Student* speed sidewalk factor
- *Brilliant Student* speed grass factor
- Initial *Excuse Generator* strength points
- Initial *Excuse Generator* excuse creation rate
- *Excuse Generator* creation acceleration
- Initial *Whining Spinner* strength points
- Initial *Whining Spinner* twine creation rate
- *Whining Spinner* creation acceleration
- The amount of time that can elapse before a *Time Tick* is no longer considered valid
- Increase in *Zombie Professor* strength points on consuming the brain of a *Brilliant Student*
- Increase in *Zombie Professor* strength points on consuming the brain of a *Excuse Generator*
- Increase in *Zombie Professor* strength points on consuming the brain of a *Whining Spinner*
- Delay in time between the death of a *Brilliant Student* and the creation of a *Zombie Professor* in its place
- *Time-Ticks-to-Strength-Points* conversion ratio.