CS4344 PROJECT
NANOWAR

Team Members:

Le Hoang Quyen	**A0110278M**	**lehoangq@gmail.com**
Zhong Qing	**A0110279L**	**a0110279@nus.edu.sg**
Peng Jun	**A0077865L**	**jasonpeng.boy@gmail.com**

I. The game
1.1. Sysnopsis

Somewhere inside a tiny world which cannot be seen by ordinary eyes, there is an eternal war.The war between a resistance force and an invading army. The resistance force called "Republic of Immune cells" try to protect their homeland (a human host) while the invader - "Brotherhood of viruses" - tirelessly find a way to invade and make the host become their home.

In this web-based game, you choose to fight for a side, cooperate with your friends and wage war against the opposite force. On one side, you will control an immune system's cell, while on the other side, you will take a role of a virus. There are several types of cells and viruses, each has its own unique skills.

1.2. Gameplay
1.2.1. Game menu:

This game allows players to play on virus or cell side and fight against each other team.

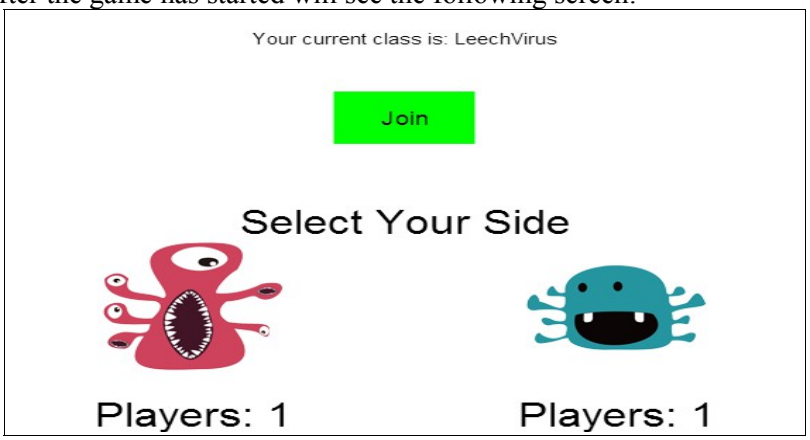
Since the current implemented version only allows one game world. The first player connects to the server will be treated as the "host" player. On his screen, there will be a "Start" button like the following figure:

The figure above shows the screen of a "host" player.

On the other hand, every players connecting after the "host" player will be treated as "guest" player and will need to wait for "host" to click "Start" to enter the game world together with him.

The figure above shows the screen of a "guest" player.

In addition, this game also allows players to enter the game even after it has started before. Any player connecting to server after the game has started will see the following screen:



The figure above shows the screen of a "guest" player after the game has started.

The "Join" button allows late arriving players to enter the game world.

1.2.2. In-game:

Player can move his character by clicking anywhere on the map to choose the destination, his character will then automatically starts moving to that destination.

There are currently 2 types of characters: the red one is called "Leech Virus" and the blue one is called "Warror Cell". Each has its own 2 unique skills:

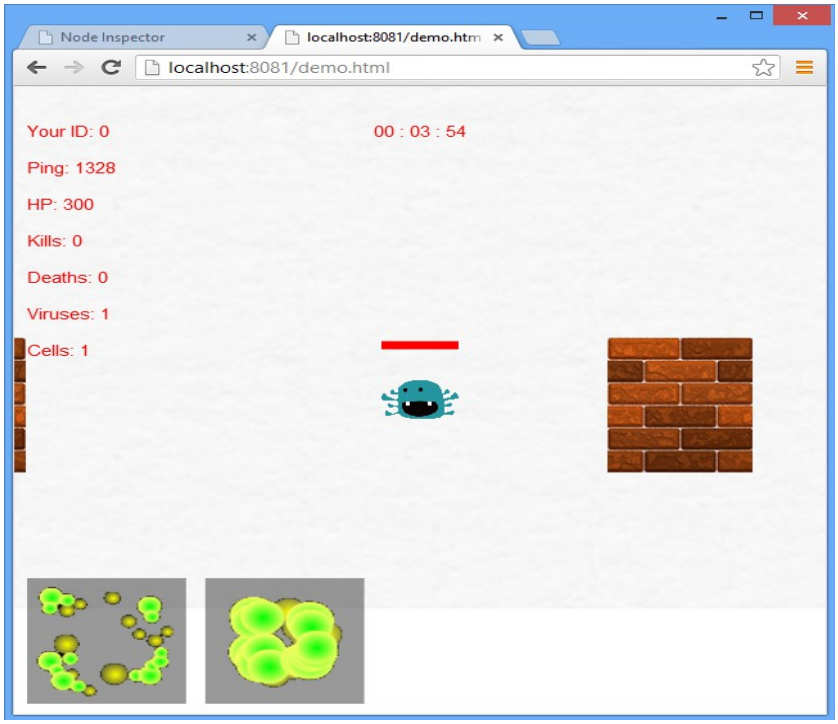
+ To use first skill, player can click a target to attack or hold down "1" key and click anywhere on the map. Though, even if holding "1" key, the 1st skill still needs a target. The "1" key is just for preventing player's character from accidently moving to clicking position instead of attacking as he expected.

+ To use second skill, player can hold "2" key and click anywhere on the map. This skill is area of effect skill, which means it doesn't need any target.

+ Smart phones users can use 1st and 2nd skill by holding down the respective skill icons which are in the bottom left corner of the screen.

There are some power-up items appearing randomly in-game. Such items can heal the player's character's health points. The power-up items for cells are medicines, while black cells are power-up items for viruses (since viruses can eat them).

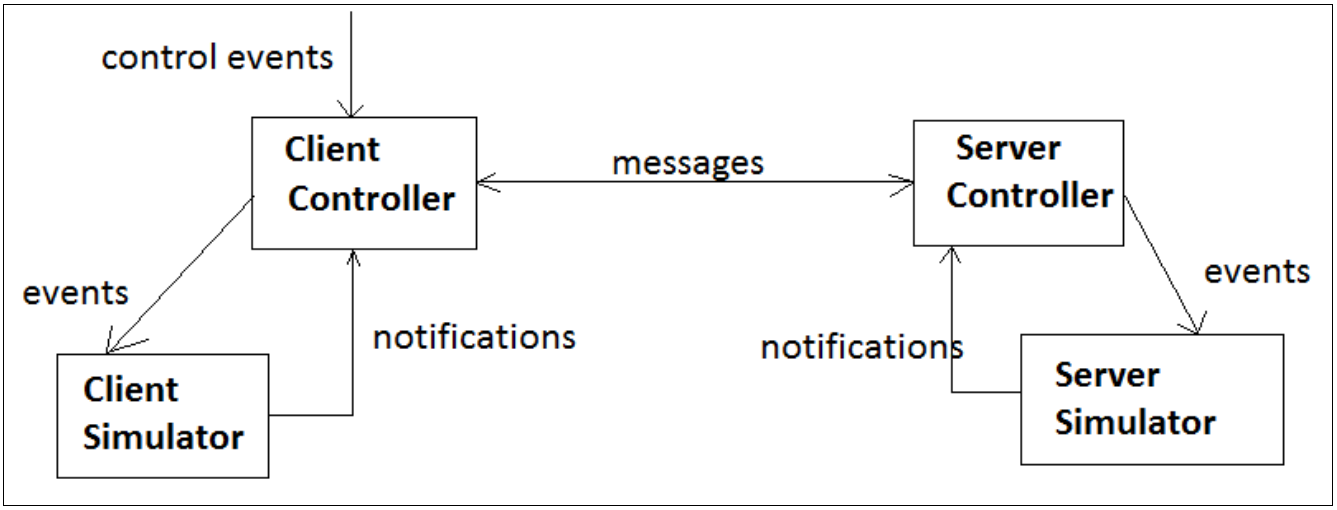
The game will last for 5 minutes, after that everyone will need to go back to the game menu and restart the game again.



The in-game screenshot.

II. Implementation

2.1. Overall architecture



The architecture consists of 4 core modules:

1. Server Controller: the interface to communicate with clients. It receives the messages from clients and then converts the messages to events before pushing to Server Simulator's events queue. Server Controller also manages a list of players that connected to the game.

2. Client Controller: the interface to communicate with server. Similar to Server Controller, it collects the messages from server as well as control events (such as mouse, keyboard inputs) then push the corresponding events to Client Simulator's events queue. Client Controller manages the information of the player and his character.

3. Server Simulator: this module's main purpose are collecting the events and simulating the game states on server side. If there are some events happen while the game is being simulated, such as an entity dies, the Simulator will notify Server Cotroller so that the Controller can notify clients if needed. Unlike Server Controller, Simulator doesn't know about players, it just manages a list of entities in game, such as characters, effects, bullets, and so on. Hence, as described above, whenever a character dies for example, it has to notify Server Controller so that Server Controller can know whose character has been died and do the appropiate works.

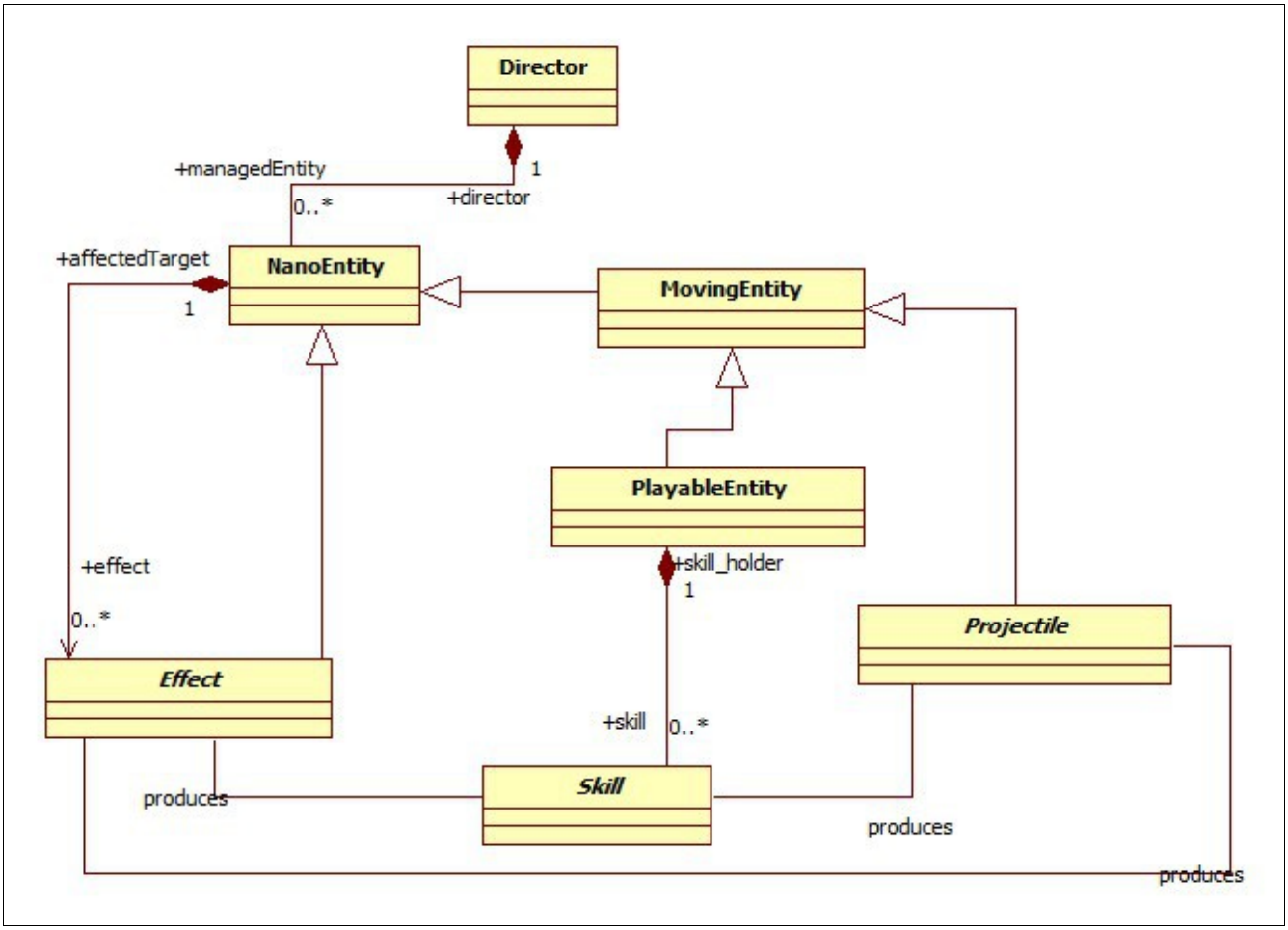
4. Client Simulator: similar role as Server Simulator, simulating the game states on client-side. Only difference is that it will not simulate some important events such as reducing the health of some characters for example, these events must be simulated on server- side.

2.2. Detailed architecture

2.2.1. Controller modules:

The Server and Client Controller modules consist of Server and Client classes respectively. They commnicate with each other via sockets and communicate with Simulator modules via Director classes, which will be introduced later. Almost every network techniques are implements in these modules.

2.2.2. Simulator modules:



The above figure describes the basic class architecture of a Simulator module.

The basic architecture of the Server and Client Simulator are the same. In fact, they share a lot of common classes. The followings are the basic classes in a Simulator module:

1. Director class: the manager of the whole simulation, it manages every entities in game including characters, bullets, effects, etc. This class is in charge of game states simulation. The Director class on client-side is a bit different from the one on server-side. However, they both are sub-classes of the DirectorBase class which has a lot of common simulation procedures like physics calculation, path finding, health points update and so on.

2. NanoEntity: the base class of nearly every classes in game.

3. MovingEntity: the base class of all classes that can "move" in game, such as bullets, characters.

4. PlayableEntity: the base class of character classes in game. Basically, the difference of this class from the above classes is that it has skills which can be used by players. Up until now, there are only 2 implemented character classes in the game: WarriorCell and LeechVirus.

5. Skill: a base class represents an ability that player can use in combat. It has an owner which is an instance of PlayableEntity. Each implemented character class will have its own set of skills. To add more skills to game later, more sub-classes of this class should be implemented. Currently, there are 4 sub-classes of this class: AcidWeapon, LifeLeech, AcidCannon, WebGun. Depends on implemented sub-class, the skill will produce projectiles (like bullets) or effects (like acid area).

6. Projectile: base class for every projectiles in game. For example, bullets.

7. Effect: base class for every effects in game. Effects can be harmful to target (like acid) or useful (like healing effect). An effect can generate another effect, such as acid area will generate acid effect on anyone in its area.

2.3. Network implementation

The implementation uses Sockjs library, thus, the transport protocol used is TCP.

2.3.1. Message structure:

Currently, the game uses simple message structure between clients and server. Each message will have an unique "type" attribute specifying which type the message belongs to, and additional attributes depends on message type. The "type" attribute of message is an integer number instead of string.

An example of message that notifies that the character needs to move to (x, y) position:

```
{
        type: 1,
        entityID: 2,
        destx: x,
        desty: y
}
```

2.3.2. Movement updates:

- From client to server:
  - Since the movement control of this game is clicking to choose the destination, every time player clicks to choose a destination, the movement will start simulating on client-side and a movement message specifying the destination will be sent to server.
  - The disadvantage is that there will be some network overheads if player clicks to change destination a lot in a short amount of time. Though, the speed of consecutive mouse clicks is reasonably not very fast.
  - The advantage is that if player doesn't change destination frequently, for example, his destination is

far away, there will be very few number of movement updates from client to server, even if along his path to the destination, he changes direction a lot. The reason is that client doesn't send updates when the character changes direction, instead the updates will be sent when the destination changes. And based on the destination, server and client will compute the path to the destination only once.

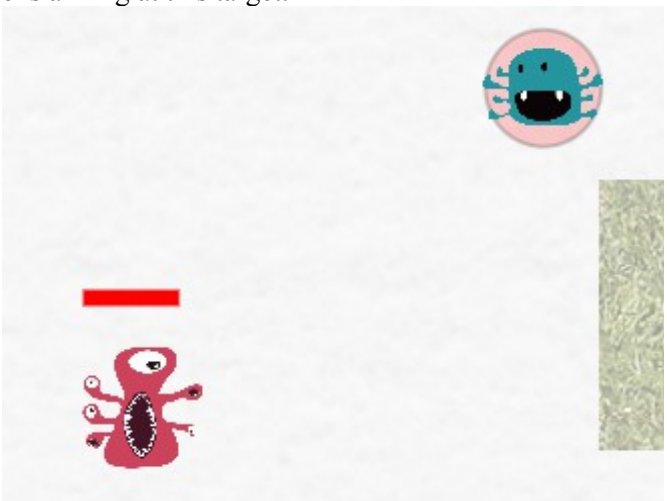
- Another advantage is that the end position of player will be consistent between clients and server.

- From server to clients:
  - An adaptive dead reckoning scheme is being applied to send movement updates of a player (denoted by A) to other players. Currently, there are 3 dead reckoning error thresholds based on 3 ranges of distance between A's character and other players'. Thus, each player will have 3 prediction versions on server.
  - For example, suppose that the 3 ranges are: [0, 10), [10, 50), [50, +∞). The respective error thresholds are: 2, 5, 10. Suppose server wants to decide whether it needs to update position of A to B or not.
    If distance between A and B is 45. Then the 2nd prediction version will be used to determine whether position of A should be sent to B or not.
  - The advantage is that if 2 player is very far away, update frequency will be very low.
- One important thing to point out is that the position used in important decisions such as whether a character is hit by a bullet or not, for instance, is the position on server side. Thus, server needs to know positions of players as accuracy as posible at any time.

2.3.3. Attacking updates:

To keep consistency between clients and server, when a player aims at another one and clicks to attack, attacking player's client side has to wait for approval from server before the actual attack takes place. This is for preventing the situation such as position of attacked target are different on attacking player's side and server side at the time the attacking click happens.

To help attacking player feel that the click command is responsive. Between the time the click happens and the time the approval from server arrives, there will be a visual trick such as an animated red mark sticking to his target to specify that he is aiming at this target.



The firgure shows the red mark indicating the red player is aiming at blue player, the attacking approval from server hasn't arrived yet.

The frequency of attacking messages from clients to server are uasually not high, since there are a waiting duration between two consecutive attacks of the same skill.

The disadvange of this strategy is that player with high network delay will have to wait for a while before his attacks take place. Considering that there is also a waiting duration for the skill to be used again, the overall waiting duration will be accumulated by the network delay.

To help improving the fairness, after the approval from server arrives, the skill's waiting duration will be reduced by an amount relative to the estimated round trip time (RTT). For example, suppose that normally the

skill's waiting duration is 800ms and the RTT is 500ms. After player receives attacking approval from server, instead of wating for 800ms, now he will only need to wait for 800 – 500 / 2 = 550ms before next attack.

2.3.4. Other updates:

As described before, the simulation on client-side lacks some important events. For example, on client-side, when an acid bullet hits a target, an acid effect that reduces target's health overtime will not be generated. Instead, that effect will need to be notified via network from server-side.

Still, the effects on client-side are mostly for visual display, they will not change their targets' health points. Thus, another update type from server called "health update" will be sent to notify how many health points of a character has been changed since the past. The health update will not be sent repeatly, instead, it will be sent when there are changes in the health points of an entity, and there is a delay between two consecutive updates.

For example, suppose the delay between 2 health updates is 300ms, within this duration, a character is hit by 2 bullets, each bullet reduces his health points by 30. The two negative health points changes will be accumulated into one "-60" health update before sending to clients. Using that way, instead of sending 2 health update messages, there is only one update.

III. Limitations and issues

1. Current system only has one game session running at a time. Future improvement should be made to support multiple game sessions. The very first draft idea about this game was supporting a lot of players playing in one single game world. However, we still don't have time and chances to test the game with many players. The multiple game sessions can be supported by allowing multiple instances of Server Controller and Server Simulator module.
2. The power-up feature has some inconsistent issues between clients and server, such as a drug moves in different directions on server-side and on client-side. It is probably due to floating point errors leading to different calculations in physics engines. We try to apply convergence to smoothly correct movements between server and clients. Still, the drugs still move pretty erratically sometimes, such as suddenly moving very fast to another place (it is not teleporting, since the convergence is being applied).
3. Some interset management (IM) strategies should be implemented, such as quadtree distance based IM. Still, adaptive dead reckoning scheme somewhat helps reducing number of updates.
4. The current renderer on client-side needs every resources (such as images, configuration files) to be fully loaded before it can start the game. Sometimes, we notice that HTTP server freezes and won't send static files to clients, leading to clients getting stuck in the white screens. It is probably due to overloaded server.
5. Even though we intended to implement two types of match: Deathmatch and Tower defence, we ended up only having enough time to finish the former one.