# CISS 350
# Tower Defense Requirements (Subject to change)

The primary goal of the project is to implement the quadtree data structure for collision detection. There are two versions of this project: Version 1 is more involved. I suggest you do version 2.

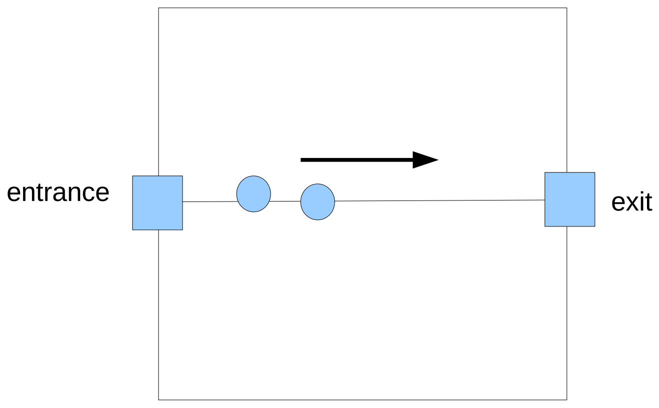
## VERSION 1

The first thing to do is to google for “tower defense” and play the game online for free. Here's one: http://www.miniclip.com/games/bloons-td-5/en/. You can read up more about this game here: http://en.wikipedia.org/wiki/Tower_defense.

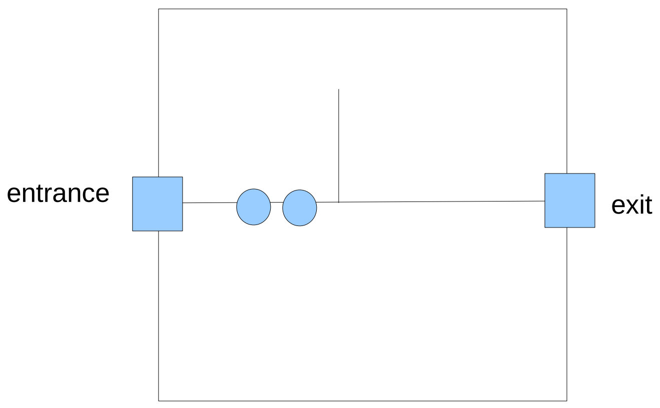
As always, if the program crashes at any point in the game play, the project grade will be zero. If you do not turn in your project by the deadline, the grade is also zero.

REQUIREMENTS

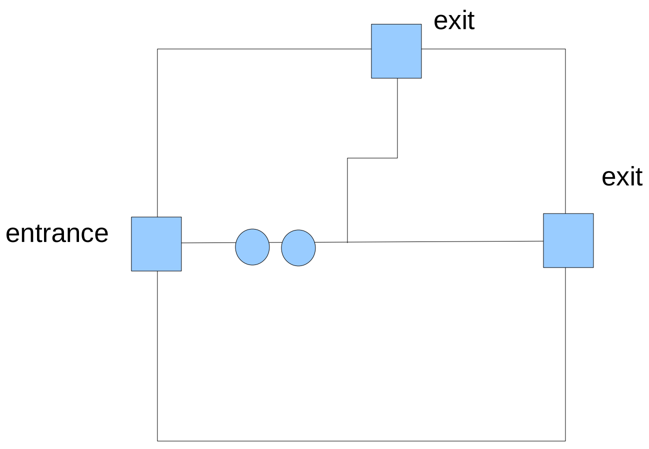
- (See simplified project option on last page.)
- When the program first runs, a welcome window is shown. The welcome window should show texts containing the name of your game and your name. Furthermore, it prints a message telling the user to press the spacebar to begin the game. If the user closes the program, the program halts.
- You have maximum freedom on designing the game. There are however several elements of the game that must be implemented.
- When the game begins, in a square area of the game window, you see one path going from left to right. There is a “balloon entrance” on the left end of the line – balloons enter the game from. On the right end of the line, there is a “balloon exit” – balloon exit the game at this point.



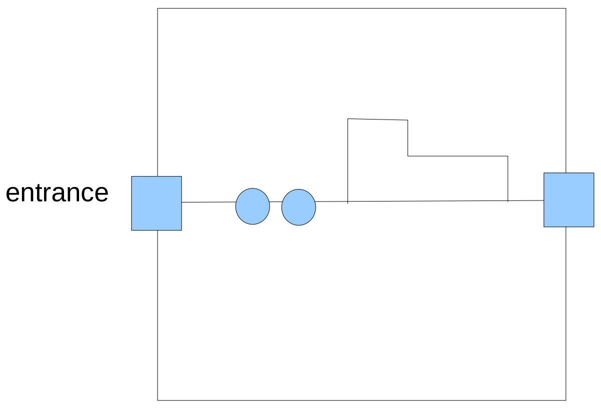
- The game play area should be a sufficient large square. It should at least be large enough to show >1000 balloons at the same time. This will ensure that you have thought through carefully on how to optimize the performance of your game.
- There is a counter in the game window that tells you how many balloons have entered the “balloon exit”. (For instance you have have a “status bar” below the game play window for that).
- The player can place towers in the game play area.
- Each tower has a range, say a radius 100. (You can have towers of different ranges.) If a balloon is within range, the tower will show with a line from the tower to the balloon.
- Each tower selects a balloon within range and fires at it. You can choose how the tower selects the balloon to destroy. For instance it can choose a random balloon in range, the closest one in range, the tower can account for the speed of the balloon, etc.
- When a bullet (or whatever weapon) from the tower collides with the balloon, the balloon and the bullet both disappear.
- Besides the above path pattern, you can have different ones, including cases where there are multiple entrances and exits.
- As the game progresses, new paths are forms. The “grow” out of exist paths. For instance here's one that's growing:



- You can either have paths growing randomly or according to some fixed pattern. I leave that to you.
- When the growing path touches the edge of the game play area, it comes a balloon exit:



- When the growing path touches a point on the path, it stops growing:



- Balloons can travel on all paths, including new ones or growing ones.
- You can have as many different type of balloons as you like, as many towers as you like, etc.

The important thing is to allow the game to run with a large number of balloons, a large number of towers, a large number of path segments. (For instance you can have a mode where the player does not die.)

- Since the paths grow, it's possible that a growing path can at some point meet a tower. I leave it to you to decide on the response to such an intersection between a growing path and a tower. When I first proposed the project, I propose that you make the path push the tower “aside”. However I have decided that another simpler option is to make the path “kill” the tower when the path touches the tower. The end result either way (or whatever other plans you have) is to make sure the tower does not intersect any path.

Now let me tell you the elements that are important:

- Your game must allow the creation of any graph. You must use an efficient data structure to represent the paths.
- You must use some form of quadtree data structure for collision detection. To show that a quadtree is used, your program must allow the display of the quadtree (say when a button is pressed). In the status bar/window of your game, display relevant statistics of your quadtree such as number of nodes, height of the tree.
- Your balloons are smart: they can pick the safest paths (i.e., ones that will avoid towers as much as possible) to arrive at exits.

The only really important thing is to use the data structures and algorithms in the above context. Therefore the only really important thing is that I want to see a graph for the paths (or whatever else you want to do such as a scene graph …) or some form of space subdivision tree for collision detection.

You are obviously not allowed to use existing code (from the web or books). You can read up in general about graphs and trees (of course!) and reading pseudocode is perfectly OK.

EXTRA CREDIT

- Games are soft real-time systems fighting for resources. Read up on priority queues and use it in your game. In your status bar/window, display statistics on the queue including average time usage for different type of objects.

You can suggest other features including those for extra credit.

SUBMISSION

- You must turn in your project by Thursday 5PM of the final exam week. You must bring your project to me personally so that I can copy your whole project folder to my desktop. You can test run the project on my desktop to verify that you have turned in the correct version.
- If you want to do a CS presentation during finals week (this is usually Thursday 9AM-12PM during the finals week), you have to finish the project by Wednesday and meet me in my office to do a demo. The seniors will be presenting their capstone projects and they will go first. If there's not enough time for non-senior projects, I will pick who to go.

## Quadtree

The node of a quadtree store points. (The 3d analog of the quadtree is called the octree.) See my notes on trees. You can also use google to find out more about quadtrees. It's essentially a 4-ary tree. Each node contains points in rectangular region of a 2d space. When the number of points reach a maximum limit, the rectangular region that the node keeps track on is equally divided into 4 smaller rectangles. The points in the original rectangle is now placed into the smaller rectangles. The purpose of doing this is a minimize the number of computations of collisions between objects in a game. If you have 1000 objects moving around in a computer window, then you need to look at roughly (1000^2)/2 = 500000 pairs of  objects and see if they collide. If the 1000 are equally distributed into 4 smaller regions, then the collision is between the pairs of objects in each room, which means roughly that there are (250^2)/2 * 4 = 12500 pairs of objects fo analyze.

Although you can use any online material to study the quadtree, you must write your own code, otherwise it will be considered plagiarism.

## VERSION 2: Simplified Project

**Demo:**
**https://photos.app.goo.gl/Z5dLy99RLMyg4wET9 (Zach)**
**https://drive.google.com/file/d/1qjDhGvNlNONMwcqvjkVBgKh0Mq3bsT5s/view?usp=sharing (Sushil)**
**https://photos.app.goo.gl/HPZUqzi3hoBubsETA (Karissa)**

Write a simulation of 1000 circles in a window. The circles moves and collides with each other. In your simulation, draw the boundary of the region for each node in the quadtree. When a circle collides with another, change it's color. You just need two colors for each circle – toggle the color on collision. You must have a key that changes between collision by pairwise check for all pairs of circles and collision using a quadtree. Choosing collision without quadtree should significantly slow down your simulation. If the number of circles is too small, increase it so that a slow-down is observable.