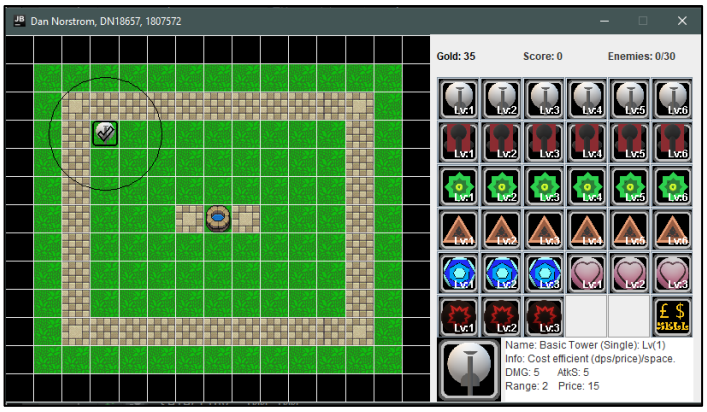
A BASIC TOWER DEFENCE GAME MADE USING SWING

Dan Norstrom (DN18657, 1807572)

CE203

UNIVERSITY OF ESSEX

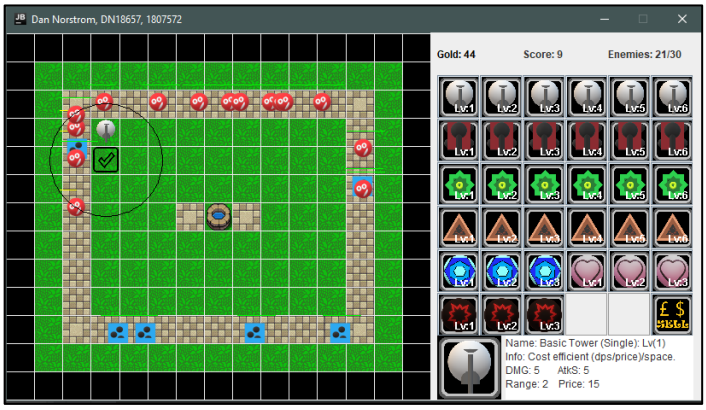
15/12/2019



Proof of a): The different towers in the shop on the right-hand side is extensions of the abstract class ***"Towers"***. They can be placed into the grid on the left-hand side by using either the mouse or keyboard input. They can be placed on any grid that has grass on it but cannot be placed on the pavement, border or an already existing tower.

When placing a tower or selecting an already existing tower the game shows you how far the attacks may reach (its range) by drawing an oval.

There is also the abstract class *"Monsters"* that the tower subclasses interact with. These monsters spawn in numbers every 10 seconds and traverses the pavement. The tower subclasses attack the monster subclasses.



```
static ArrayList<Monster> monster_list = new ArrayList
static ArrayList<Tower> tower_list = new ArrayList


public abstract class Tower {

    private int xC;
    private int yC;
    private String type;
    public abstract String getType();
    public abstract void setTarget(Monster target);
    public abstract Monster getTarget();

    public abstract String getName();
    public abstract String getInfo();
    public abstract ImageIcon getIcon();

    public abstract int getX();
    public abstract int getY();
    public abstract void setX(int xC);
    public abstract void setY(int yC);

    public abstract int getPrice();

    public abstract int getATS();
    public abstract void setATS(double ats);
    public abstract void setDMG(double dmg);

    public abstract int getRange();
    public abstract int getDMG();
    public abstract void clearTarget();

    private Monster target;

    public abstract int getLVL();

    public abstract Tower copy();

}


public abstract class Monster {

    private int xC;
    private int yC;
    public abstract String getName();
    public abstract String getInfo();
    public abstract ImageIcon getIcon();

    public abstract int getX();
    public abstract int getY();
    public abstract int getV();
    public abstract void setX(int xC);
    public abstract void setY(int yC);
    public abstract void setV(int velocity);

    public abstract int getMaxHP();

    public abstract int getG();

    public abstract int getHP();
    public abstract void setHP(int xC);

    public abstract int getVC();

}
```

Proof of b) Monsters traverse the grid by a change in coordinates controlled by the game in correlation to the classes **setY()** and **setX()** methods.

The **getIcon()** enables the game to draw this class into the game. When a monster is generated by the class **"MonsterSpawn"** it's added to the collection **"monster_list"**. The game renders and updates all monsters in arbitrary positions from the **"monster_list".**

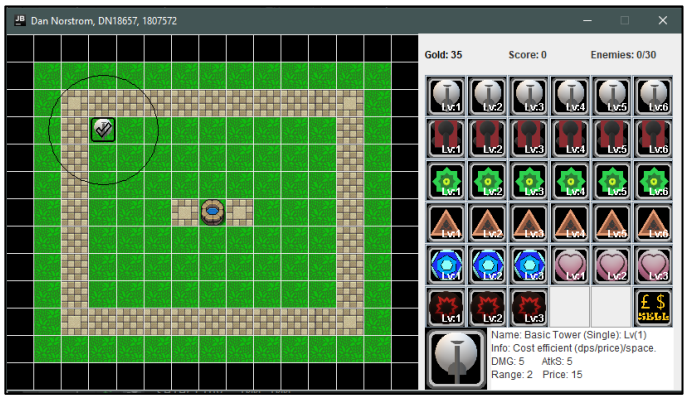
Towers are placed based on the current selected grid with the methods **setX()** and **setY().**

After a grid has been selected and a tower has been bought, it fetches the current grid coordinates and adds a tower with those coordinates into the collection **"tower_list".** The game renders all the towers in the **"tower_list"** with the help of **getIcon()** and paints the information of each tower with **getInfo()** and **getName().**

Encapsulated classes/extension: all different monster classes such as **"Monster_YellowTriangle"** and **"Monster_WaveBoss"** extends the abstract class Monsters.

While all tower classes such as **"Tower_MultiStar"** and **"Tower_Basic"** extends the abstract class Towers.

All of these objects are held in their individual collections **"monster_list"** and **"tower_list"** as long as they exist in the game.

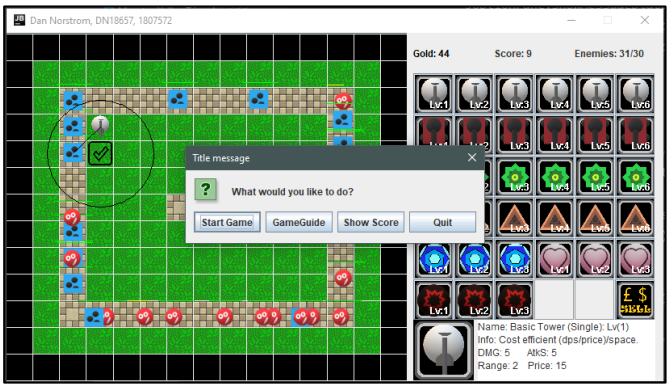


Proof of c): One may use both the mouse and keyboard to play this Tower defence game.

Using the mouse to select a grid and thereafter double clicking a tower to buy it. One can also sell towers or click towers once to see that towers information.

Likewise, one may move the position of the grid using the keyboardlistener. Arrow keys moves the position on the grid while a combination of (Q,W,E,A,S,D)(x-coord) followed by (1,2,3,4,5,6)(y-coord) buys towers in the shop menu.

These listeners work in tandem and any combination of mouse/keyboard may be used!



roof of d): Before and after games the interface menu will be shown. This screen allows the players to see their top 5 score as requested. In addition, after every game the current games score will always be shown.

At the bottom left is a shorted version of the class responsible for the score.

```
public class ReportScore {

static File file = new File("./Top5Scores.txt");

public static void addScore(String score)
{…}

public static ArrayList<Integer>ReturnScore()
{…}

}
```

Game Architecture:

To no surprise the main class in this tower defence game is “Game.java”. All other classes have a certain amount of coupling to this class while retaining a good amount of cohesion by isolating different concepts in new classes. There are 3 runnable classes that haven’t been put into their own files because of how high cohesion they have with game.java. Therefore, based on OOP design, I’ve kept these integrated. Otherwise they would have a very high degree of coupling which we want to avoid.

Game.java: initializes itself using its GameCanvas() method. This creates the game grid, the shop grid, all menus and JComponents (except Monster_Panel which is generated). It also adds actionlisteners and some basic paint elements to grid buttons.

::BackgroundPaint() is a helper function that assists the canvas with button painting.

::Class RunRender is a Runnable to be instantiated with a thread. Responsible for rendering and updating monsters locations, gold, score, enemy count. It shows if you’re allowed to build on the current grid with different icons and removes dead monsters from the monster_list.
Key concept: Repaints monster_panel made in GameCanvas() to render monster movement and tower attacks (lines).

::Class RunWave: Is a Runnable to be instantiated with a thread. Responsible for generating monster waves every 10 seconds. It couples well with MonsterSpawn to add new monsters to the active monster_list.

::Class RunAttack: Is a Runnable to be instantiated with a thread. Responsible for calculating towers attack speed, range and what their current target is. It also deals damage to monsters attacked by towers by calling monster.setHP(tower.getDMG()) on targeted monsters in monster_list.

::Class threadMenu Is responsible for showing the games interface menu and initializing runnables when a new game starts. A new game starts if newGame=true and it ends if endGame=true. Runnables initialized here all ends if endgame is true. (this happens when the RunRender detects more then 30 monsters alive on the map)

MonsterSpawn.java: Takes two integer lists from RunWave and generates monsters from Monsters subclasses and sends them to the running games monster_list.

Problems & Improvements:

A recurring problem was how to render multiple monsters on one panel in swing without glitching and render issues. The glitching was solved by splitting the work of different timers into threads working in tandem. The rendering issues was solved by adding a click-trough background of the grid to the monster_panel followed by different layer levels on a JLayeredPane instead of a Jpanel.

The game uses a decent amount of resources (60-100mb ram). This could probably be minimalized by using more primitive datastructures and reworking the architecture.

The software has some parts that could be reusable. Especially the ReportScore.java and MonsterSpawn.java concepts. I'm quite happy with how the threadMenu initializes threads but not so happy about the Boolean being used to close them or how its nested together with a JOptionMenu. There are elements in the threads that have high cohesion with other parts of the code (such as the endgame Boolean), this could hopefully be improved with a more intricate thread handler class.

Final Words:

Overall I've managed to get where I wanted and the game plays beautifully. The gameguide option in the interface will hopefully aid in understanding the basic game mechanics. Making a tower defence game in swing was a learning experience and I believe that my game architecture requires a lot of hashing out. *I hope you enjoy playing the game for a few minutes and thank you for reading!*

Ps. If you wanna try some better towers edit `gold = 50;` *in game.java row 901 :D*