BotWorld (a.k.a. RatBots 16)

*Instructions and Rules*

# Overview –

What is BotWorld? BotWorld is a computer programming challenge. You will use your programming skills to write the code to control a Bot that will be placed in a computer-generated arena (the BotWorld). This year the goal for your Bot is to acquire the most Prizes (details about this are shown later in this guide.) You will need to apply all of your Java programming skills to design (and redesign, and redesign…) a Bot that can achieve the goals within an arena as efficiently as possible.

Competitions against other Bots.

There will be many different competitions and methods of judging the Bots, and thus some may be designed to win a certain type of competition and some to win others. A few super-Bots may be able to sweep the competitions! You may *enter up to three* different Bots into the competitions. If you choose to submit multiple entries, please make sure that they are substantially different from each other. Each of your entries will compete in all of the ‘events’.

LIST OF EVENTS:

1. **The Bot League** – Each Bot will compete against each of the other Bots in a head to head competition of 100 rounds. The Bot that wins the most rounds will win the match. The Bot with the best overall win-loss record will win the competition. In the event of ties, total rounds won will be the only tiebreaker.
2. **The Singles Competition** – Each Bot will be individually placed in the maze for 100 rounds. The Bot that acquires the highest total score will win this competition.
3. **The Battle Royale** – By random draw, the Bots in groups of up to nine will be placed in the maze for 100 round competitions. The top three or four Bots in rounds won in each match will advance to the next matches. Once the field has been narrowed to nine or fewer finalists, a 100 round live computer-cast final will decide the winner! (The exact structure of this competition will be decided once the number of entrants is known.)
4. The following awards will be awarded via judging. **Most Efficient** solution(success with small code size)and the **Most Entertaining** solution.

Extra rules which apply to all events:

Each round will end when 500 moves are completed. The Bot with the highest score wins.

In a competition, the order that the Bots are selected to act is random each move.

In a competition, if multiple rats have the same high score when the round ends, each will be awarded a round win.

If your Bot function crashes at any time, it will be eliminated from the competition.

If your chooseAction function takes too long to run (an average of over 10ms/move on a classroom computer or as determined by the committee) it could be disqualified from competitions.

Please remove ALL System.out.print type statements from your submitted Bots!

Deadlines for submissions will be posted on the website.

Please check the website for updates and clarifications.

The Arena

* The arena is a 21 by 21 grid. The top left corner has a Location of (0,0).
* There is a fixed setup of Blocks in the arena at the start of each match (shown below.)
* There will always be a Block in the center of the arena. Each Bot will be started at a random location bordering this Block. This begins any match on fairly even ground.
* There are three different types of Prizes:
* At the start of the game there will be one golden Prize worth 500 points. Every 100th turn a new golden Prize will be placed in the arena. These special golden Prizes will only appear within the ‘rooms’ on each edge of the arena (on the spaces marked with asterisks.) No other Prizes will ever appear in these ‘rooms’.
* After the first turn and every 10th turn there after a new red Prize will appear in a random location. No location will be chosen that is within 10 spaces of any Bot. If no empty space exists that meets these criteria, no Prize will be generated. These red Prizes are worth 10 points each.
* After a Block is constructed a new blue Prize will be constructed with the same location restrictions as red Prizes. Blue Prizes are worth 25 points each.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | \* |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | \* |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | \* |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  | \* |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

How to write the BotBrain for your Bot

Begin with a Java class that **extends the BotBrain class**.

You must override the chooseAction( ) and *optionally* the initforRound( ) methods.

**public void initForRound()**

Whenever a new round begins, the BotWorld engine will call your BotBrain’s initforRound( ) method. Use this method to initialize any variables that need to be reset at the start of a round.

**public int chooseAction()**

The BotWorld engine will call your BotBrain’s chooseAction( ) method whenever it is your turn to move.

Your BotBrain’s chooseAction( ) method will **return an integer** value that corresponds to the next move your Bot will make. **There are many possible moves you may choose from:**

**MOVE** – if you return a value of 0, 90, 180 or 270, your Bot will attempt to move in that direction. (0=NORTH, 90=EAST, 180=SOUTH, 270=WEST.) If a Block or another Bot blocks your path to move as requested, the Bot will not move this turn.

**DART** – if you return a value of 1000, 1090, 1180 or 1270, your Bot will attempt to ‘dart’ in that direction. (1000=NORTH, 1090=EAST, 1180=SOUTH, 1270=WEST.) A ‘dart’ move will move the Bot as far as possible in that direction (going until it hits a border, Block or another Bot.) Prizes along the darting route are NOT gained. (You only gain Prizes that you end your turn on.)

**BLOCK** – if you return a value of 2000, 2090, 2180 or 2270, your Bot will attempt to build a Block on the space in that direction. (2000=NORTH, 2090=EAST, 2180=SOUTH, 2270=WEST.) Blocks built in this way will stay in the arena for the entire round. Blocks cannot be placed onto other Bots. They can be placed onto Prizes resulting in the Prize disappearing.

**WALL** – if you return a value of 3000, 3090, 3180 or 3270, your Bot will attempt to build a wall of temporary Blocks seven spaces wide, centered on the space in that direction. (3000=NORTH, 3090=EAST, 3180=SOUTH, 3270=WEST.) Blocks built in this way will disappear after ten moves. Blocks cannot be placed onto other Bots. They can be placed onto Prizes resulting in the Prize disappearing.

**REST** – if you return any number not specified above, your Bot will not move this turn.

**The following constants are defined in the Bot class for convenience.**

public static final int REST = -1;

public static final int MOVE\_NORTH = 0;

public static final int MOVE\_EAST = 90;

public static final int MOVE\_SOUTH = 180;

public static final int MOVE\_WEST = 270;

public static final int DART\_NORTH = 1000;

public static final int DART\_EAST = 1090;

public static final int DART\_SOUTH = 1180;

public static final int DART\_WEST = 1270;

public static final int BLOCK\_NORTH = 2000;

public static final int BLOCK\_EAST = 2090;

public static final int BLOCK\_SOUTH = 2180;

public static final int BLOCK\_WEST = 2270;

public static final int WALL\_NORTH = 3000;

public static final int WALL\_EAST = 3090;

public static final int WALL\_SOUTH = 3180;

public static final int WALL\_WEST = 3270;

Each BotBrain, by extending the BotBrain class, has access to many ‘sensing’ methods that will tell you what your Bot is ‘seeing’ in the maze. A summary of these methods is given on the next page. Please look at the demo Bots (included with the code) to get an idea of where to start coding.

### Summary of methods in the **BotBrain** class

|  |  |
| --- | --- |
|  | |
| **Modifier and Type** | **Method and Description** |
| int | **getCol**()  Gets the x coordinate within the maze of this Bot. |
| int | **getRow**()  Gets the y coordinate within the maze of this Bot. |
| Location | **getLocation()**  Gets the Location (row,column) of this Bot. |
| GameObject[][] | **getArena**()  Gets a 2-dimensional array (row,column) that contains the GameObject at each space in the arena. GameObject is the parent class for Bots, Blocks and Prizes.  Will return null for spaces that do not contain a Block, Bot or Prize. |
| int | **getScore**()  Gets the current score in this round for this Bot. |
| int | **getBestScore**()  Gets the current best score in this round among all Rats in this match |
| int | **getRoundsWon**()  Gets the number of rounds won in the match by this Bot. |
| int | **getMoveNumber**()  Gets the current move number within this round. |
| int | **getRoundNumber**()  Gets the current round number within this match. |
| boolean | **canMove**(int dir)  Determines if this Bot canMove one space in the specified direction. |
|  |  |
|  |  |
| java.lang.String | **getName**()  Gets the name of this Bot. |
| void | **setName**(java.lang.String in)  Sets the name of this Bot. |
| java.awt.Color | **getPreferredColor** ()  Sets the name of this Bot. |
| void | **setPreferredColor**(java.awt.Color in)  Sets the preferred color of this Bot. |
|  |  |

|  |  |
| --- | --- |
|  | These methods are provided in the **Bot** class. |
|  |  |
| **Modifier and Type** | **Method and Description** |
|  |  |
| int | **getCol**()  Gets the column (x coordinate) within the maze of this Bot. |
| int | **getRow**()  Gets the row (y coordinate) within the maze of this Bot. |
| int | **getScore**()  Gets the score of this Bot. |
| java.lang.String | **toString**()  Creates a string that describes this Bot. |
|  | These methods are provided in the **Location** class. |
|  |  |
| **Modifier and Type** | **Method and Description** |
|  |  |
| int | **getCol**()  Gets the column (x coordinate) within the maze of this Bot. |
| int | **getRow**()  Gets the row (y coordinate) within the maze of this Bot. |
| Location | **getAdjacentLocation**(int dir)  Gets the score of this Bot. |
| boolean | **isValidLocation**()  Gets the score of this Bot. |
| java.lang.String | **toString**()  Creates a string that describes this Location. |

|  |  |
| --- | --- |
|  | These methods are provided in the **Prize** class. |
|  |  |
| **Modifier and Type** | **Method and Description** |
|  |  |
|  |  |
| int | **getCol**()  Gets the column (x coordinate) within the maze of this Prize. |
| int | **getRow**()  Gets the row (y coordinate) within the maze of this Prize. |
| int | **getValue**()  Gets point value of this Prize. |
| java.lang.String | **toString**()  Creates a string that describes this Prize. |
|  |  |
|  | These methods are provided in the **Block** class. |
|  |  |
| **Modifier and Type** | **Method and Description** |
|  |  |
|  |  |
| int | **getCol**()  Gets the column (x coordinate) within the maze of this Block. |
| int | **getRow**()  Gets the row (y coordinate) within the maze of this Block. |
| int | **getDuration**()  Gets the number of turns before this Block disappears. |
| java.lang.String | **toString**()  Creates a string that describes this Block. |
|  |  |
|  |  |
|  |  |
|  |  |

How to use the BotWorld programs

There are two primary ways that you can enter your Bot into the maze.

1. If you place your BotBrain’s code into the brain package within the project it will automatically be loaded into the menu when you run the project. You can select to add your BotBrain from the *Add Bots* menu.
2. You can directly load a Bot into the maze by changing the BotWorldRunner file. Instructions are given in the comments of that file. When you are testing a Bot repeatedly, this can save the time of loading it from the menu.

## Troubleshooting a Bot – suggestions

Once you get the BotWorld program running, if your Bot doesn’t move, it’s probably your fault. Check your algorithms. (However, please let us know of any bugs you find.) Start simple and test your Bot after each improvement you make. Some ‘improvements’ end up negatively. Watch many trials of your Bot to get a feel for where it gets stuck, and how you could fix that.

Other notes about creating Bots:

You may write sub-functions that get called by your BotBrain’s chooseAction and initForRound functions.

Your BotBrain will NOT have a main function. The main function you will use is within the BotWorld project.

Bot names should be creative, descriptive, appropriate and less than 15 characters long.

Each BotBrain should have its own .java file.

You don’t have to write any graphics functions. They are all taken care of in the BotWorld project.

There are many BotBrain examples provided with the code. These will give you a starting point for designing your BotBrain. Feel free to use any of the code found in the examples, however you should give credit for any ideas you get from these in your comments.

A brief history of Bots

2000-01 – First Bots competition. The code was written using a graphics library created by student Danny Wilhelm. The goal was simple – escape the maze as quickly as possible.

2002 – Still coding with C++. Rules changed to make the game an acquisition of territory game.

2003 – First year with Java!

2004-2011 – Many different variations of the game over the years including: Capture the flag, playing in teams, power-ups, managing energy levels, stunning your opponents, chasing down (or avoiding) computer created robots, paintballs, teleportation, variations on acquiring territory, and mazes full of cheese.

2012 – Began using the Grid World platform for the game.

2013 – The goal was trying to have the longest tail. Class loader code added by student Patrick Angle.

2014 – The goal was based on finding the cheese, with bonus points for aged cheese and cheese outside the maze. Some teleportation was involved!

2015 – Inspired by logistics within warehouses and factories. Very challenging!

2016 – Rebranded as BotWorld, Bots can ‘see’ the entire arena as they look to acquire Prizes. Useful shortcut keys in Bots

|  |  |
| --- | --- |
| F1 | Switch Arena to Challenge #1 mode. (One Prize / No Blocks) |
| F6 | Switch Arena to Normal mode. |
| F7 | Display last move chosen for each Bot |
| F8 | Change color scheme |
| F9 | Toggle Grid Lines |
| F10 | Start running in console mode |
| F11 | Stop |
| F12 | Start running at top speed |
| SPACE | Step or toggle console |
| 1 | Load the first Bot in the menu |
| 2 | Load the second Bot in the menu |
| 3 | Load the third Bot in the menu |
| 4 | Load the fourth Bot in the menu |
| 5 | Load the fifth Bot in the menu |
| T | Change the length of the tails shown. |
| R | Reset to round one and reset scores of Bots in the Maze |
| Q | Close the program |
| ESCAPE | Close the program |

While the game is stopped, you can:

* Hover the mouse over a space to get info on what is in that space.
* Right click on an object to get a list of methods for that object.
* Right click on an empty space to add an object to the maze.

You can run the game three ways:

* Step-by-step using the step key – this advance the game on move each time.
* Run at various speeds. – Look for trends in how your Bot is doing.
* Console mode – runs many rounds very quickly. See your average score!

(We always run at least 100 rounds to get a good average measure of performance because some arenas are more difficult than others due to random chance.)