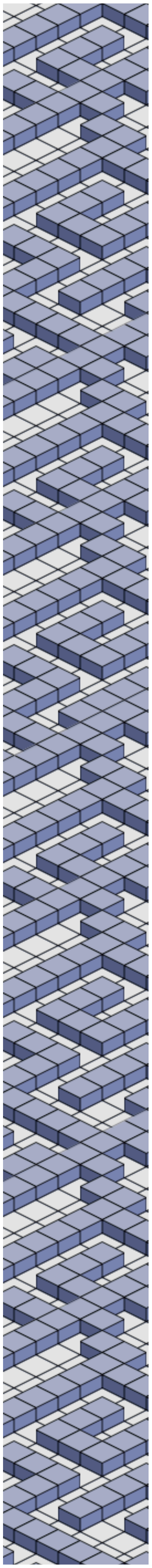


The Darwin Game 2.2
Player's Handbook

In The Darwin Game, creatures compete to control maps and race through mazes. You play by programming your own species of creature in Java, which then acts autonomously during competition. Creatures can move, sense their surroundings, and attack. A successful attack replaces its target with a new instance of the attacker, allowing creatures to reproduce. The world is rendered in isometric 3D and players can create their own creature icons. To dive right in, make a copy of Rover.java and start modifying it:

```
cp Rover.java MyGuy.java
…
run -3D ns_faceoff MyGuy Rover
```

1 / The World of Darwin

Darwin creatures exist in a grid-based world specified by a rectangular map. Different maps are available. A map square may be empty, occupied by a creature, or by an obstruction. Here are some of the common map elements:

Object	Observation	Image	Description
Wall	`type == Type.WALL`		Impassable square. Attempting to move onto this square halts but does not harm a Creature.
Apple	`type == Type.CREATURE` `classId == APPLE_CLASS_ID`		Motionless, passive Creature just waiting for you to attack it.
Hazard	`type == Type.HAZARD`		Impassable square. Attempting to move onto this square converts a Creature to an Apple as if it were attacked.
Flytrap	`type == Type.CREATURE` `classId == FLYTRAP_CLASS_ID`		A dangerous creature rooted in place. Continuously spins to the left and blindly attacks.
Treasure	`type == Type.CREATURE` `classId == TREASURE_CLASS_ID`		Attack this to complete a Maze map.
Enchanted Apple	`type == Type.CREATURE` `isEnchanted == true`		When attacked, the Creature that spawns is also enchanted.
Shrine	`shrineClassId != UNINITIALIZED_CLASS_ID`		An enchanted Creature ascends at its own species' shrine

Creatures and map squares may also have modifiers, such as being enchanted or filled with mud.

Morgan McGuire, 2008-2012
Williams College Computer Science Department

http://cs.williams.edu/~morgan/darwin/
DARWIN 2.2

Thanks to Aaron Size, Chris Warren, and Josh Szmajda for play testing. "Darwin's World" was created by Nick Parlante as concurrent assembly interpreter programming assignment in Pascal in 1992. Eric Roberts ported it to C in 1999 and Steve Freund adapted it to Java around 2000. I designed and implemented a new game based on these ideas in 2007 as the "Darwin 2.0" AI and concurrency game. "Darwin 2.1" uses a coroutine model that eliminates the need for synchronization to emphasize the AI aspects while maintaining the strategic value of efficient algorithms. "Darwin 2.2" increases complexity to admit multiple viable strategies and encourage more interesting pathfinding and metagaming.

2 / Installing Darwin

You can play the Darwin Game on any operating system. If you are playing the Darwin Game in a course, your instructor has probably given you a Java development environment and installed Darwin. If you are not in a course, or just want to work on your Creature at home, follow the instructions in this section.

Get Java

If you are on OS X, Linux, or FreeBSD then Java is probably already installed on your computer. If you are on Windows then you probably need the free Java JDK. The JDK for all platforms is available from:

http://www.oracle.com/technetwork/java/javase/downloads/jdk-7u1-download-513651.html

Get Darwin

Download the free, open source Darwin game system from:

http://cs.williams.edu/~morgan/darwin/

Unzip it and visit the directory from the command line (e.g., OS X terminal, Windows CMD). Type "run" and press enter to verify that Darwin is working.

3 / Your First Creature

Make a copy of Rover.java with a new name, like MyGuy.java. Open the file in a text editor, like Emacs, Notepad, Xcode, or Visual Studio. Rename the class to match the file and save it.

Now run your new creature inside the Darwin simulator by typing:

```
run -3D mz_1 MyGuy
```

Darwin will prompt you to compile your creature automatically. Press the play button (or one of the fast play buttons) to run the game. You can modify your creature's behavior by editing its .java file and then pressing the reload button inside the simulator.

You can also explicitly compile all .java files in your Darwin directory at the command line with:

```
compile
```

or to just compile one creature,

```
compile MyCreatureName.java
```

This shell script will issue the appropriate commands for your operating system. See README.TXT for detailed instructions on how to launch the Darwin game. Note that on Linux you may need to use a variation on these and remove the –Xdock argument.

4 / Game End

Maze

Maze map filenames are prefixed with “mz_”. They contain only one kind of creature and one Treasure. The goal on these maps is to find and attack the treasure before the time limit expires. The time limit is about 8 [virtual] seconds.

Natural Selection

Natural Selection maps are prefixed with “ns_”. They contain multiple kinds of creatures and no Treasure. The goal on these maps is to perform five *ascensions* or to become one of the most populous species. They have a time limit of about 50 (virtual) seconds.

There are *standard* tournament mazes that have further restrictions, and nonstandard ones that are useful for testing (or simply experimenting with for entertainment.)

See the Tournament section of this handbook for details of scoring Natural Selection draws and repeated trials.

5 / Creature Actions

In addition to Java commands for programming logic, creatures can perform *actions* within the world by invoking special methods on itself. Each action has a *time cost* in nanoseconds, creating tension between the time spent deciding which action to take and the time to perform that action. Using data structures effectively to reduce decision time is therefore essential. Available actions are:

Action	Cost	Description
Move Backward	700000 ns	Move backward one square. If the square is blocked, the move fails but still costs time.
Move Forward	400000	Move forward one square in the current facing direction. If that square is blocked, the move fails but still costs time.
Attack	800000	Attack the creature immediately in front of this one. If there is no creature of a different species present, then the attack fails but still costs time*. If the attack succeeds the target is replaced with new instance of this creature facing in the opposite direction. ** An enchanted creature attacking another member of its own species passes the enchantment to the target.*
Observe	200000	Return a description of the squares along the facing direction up to and including the first non-empty square.
Turn Left	600000	Rotate 90-degrees counter-clockwise.
Turn Right	600000	Rotate 90-degrees clockwise.
Delay	100000	Pass this turn. The same creature may receive its next turn immediately.
Emit Pheromone	200000	Leave a mark on the map (experimental)
Mud Penalty	+500000	Moving, turning, or attacking from a square containing mud costs more
Ascend	0	An enchanted creature moving onto its own class' Shrine ascends immediately and is replaced by a non-enchanted creature.

All creatures in the world take *turns*. When a creature performs an action, the effect occurs immediately and then the creature's turn ends (except that the result of an observe action is valid as of the beginning of the next turn). The run method need not return—on the creature's next turn, its program will continue execution immediately after the previous action's invocation. A *nonresponsive* creature that takes no action for ½ second has "stopped breathing" and is converted into an Apple.

The simulator tracks the total time that each creature has spent on its previous turns. The creature that has spent the least amount of accumulated time takes the next turn. When a new creature spawns as a result of a successful attack, it inherits the total time of the creature that created it.

Note that only one creature is active at a time. There is no need to synchronize access to data structures, and a creature that makes decisions within a reasonable amount of time can assume that its code executes without interruption until it takes an action.

Creatures are only allowed to invoke the action methods on themselves. They may not invoke action methods on other members of their species for which they have obtained pointers. The simulator enforces this and other rules intended to promote fairness. *In general, any code that the simulator permits is legal*; see the Tournament rules for exceptions.

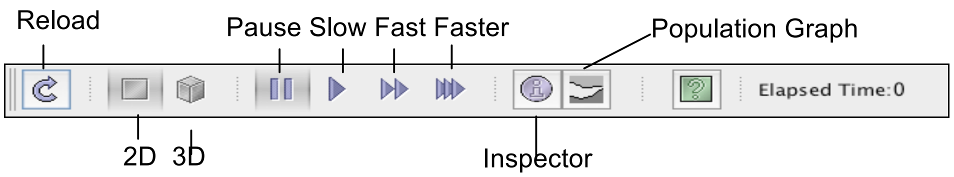
6 / The Darwin GUI

The Darwin class runs the Simulator within a GUI. Its command line arguments are the name of the map and the Creature classes with which to populate it. To launch it, use the run shell script. For example,

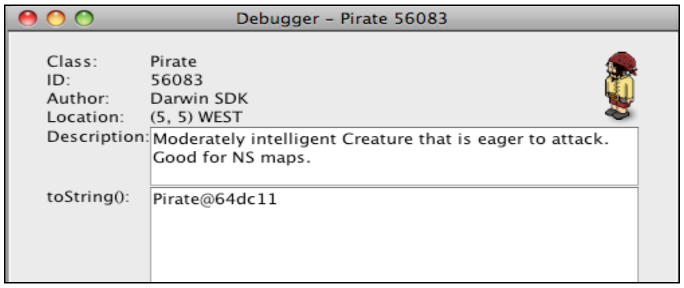
```
run ns_faceoff Rover Pirate
```

launches the simulator on the "Faceoff!" map with Rovers competing against Pirates.

The GUI always begins paused. Press one of the three speed buttons to begin simulation. The speed of simulation can be changed (or paused again) during play. The map view can be switched from 2D (good for debugging) to 3D (good for watching matches) using the gray square and cube icons.



In 2D view mode, click on any Creature to view it in the Darwin Inspector. This shows information about the Creature that updates in real time, including the current value of its toString method. **Override Creature.toString** and use the debugger to inspect the internal state as it moves through the map.



Debugger – Pirate 56083

Class: Pirate
ID: 56083
Author: Darwin SDK
Location: (5, 5) WEST
Description: Moderately intelligent Creature that is eager to attack. Good for NS maps.
toString(): Pirate@64dc11

The Reload button not only restarts the simulation, it also reloads your Creature files from the .class files. This means that if you change your Creature and recompile it, you do not need to restart the simulator. Just press Reload, as if in a web browser, and you'll get the newest version. **Darwin will even prompt and recompile your .java source file** if the class is out of date!

Darwin security prohibits Creatures from accessing the file system, the reflection API, and other Java features that could allow them to gain unfair information or interfere with the normal operation of the simulation and other creatures. The GUI (and Simulator, if you are running it directly from your own framework) allow security to be disabled. For the GUI, you can do this with the –nosecurity command line argument. This is sometimes useful when debugging your creature, and is very useful if you'd like **to "train" your creature using data files**. To run in competition you will have to embed the contents of your data files in your class, however. For example, as a private static String.

7 / The Creature API

Creatures all subclass Creature, which provides a set of protected and public methods that enable the Creature to interact with the world. Creatures must either provide a public constructor of no arguments or have no constructor. The constructor cannot make the creature take an action.

A Creature's run method executes when it is inserted into the world. When the run method ends the creature can take no further actions (it remains in the world, however). Therefore most Creatures have an intentionally infinite loop in their run method to allow them to continue taking actions.

If a Creature is successfully converted to another species, then it is removed from the world but continues executing. The isAlive() method for a converted creature returns false. If a creature that has been converted attempts to take an action, then a ConvertedError is thrown. Most Creatures catch this error and then allow their run method to terminate.

See http://cs.williams.edu/~morgan/darwin/ for the full Creature API.

Below is a sample of the code for a very simple Creature called a Rover. It moves until obstructed and then attacks the obstruction and turns to the left. It is surprisingly effective, but is unable to deal with hazards because it never looks before moving. The gray code is boilerplate common to every Creature. The bold black code in the center is the logic unique to the Rover.

```
public class Rover extends Creature {
    public void run() {
         while (true) {
              if (! moveForward()) {
                     attack();
                     turnLeft();
              }
         }
    }
}
```

Creature positions are specified using java.awt.Point, which you will need to import at the top of your class to perform any useful operations on positions. Note the helper methods on Creature and Direction that operate on Points and Observations.

The API uses Java enum types to specify Directions and creature Types. Enum types can generally be treated as constants, however they also provide useful utility methods. The following (nonsense) code demonstrates uses of the Direction enum.

```
Direction d = Direction.NORTH;

if (d == Direction.SOUTH) { … }

d = d.left();

Point p = new Point(3, 4);
```

```
p = Direction.forward(p);

switch (d) {
  case NORTH:
       ...
       break;
  case EAST:
       ...
}
```

8 / Tournament Rules

A creature for a tournament must be submitted in a zip file whose name is the name of the creature **followed by the author's initials** in all caps, e.g., WolfMM.zip. Inside the zipfile must be a single .java source file whose name matches that of the zipfile (e.g., WolfMM.java), and the four icons described previously with compatible names (e.g., WolfMM-E.png, etc.)

Standard Maze Tournaments have exactly two creature instances: one treasure and one competitor. *Standard Natural Selection* tournaments have four competitors (each starting with the same number of instances) and enchanted Apples. They may also contain FlyTraps and non-enchanted Apples. Unofficial Natural Selection tournaments may have arbitrary configurations, such as 1-on-1 (which was the format until 2012).

Intentionally creating a denial of service condition is grounds for, but may not result in, disqualification. A normal infinite loop is *not* a denial of service, since creatures are run on their own threads. Note that the simulator is hardened to prevent *accidental* stack overflow or allocating all heap memory.

Creatures in a tournament *are* permitted to attack the JVM and running simulator to attempt to gain access to restricted information or corrupt the simulation state to their advantage. However, violating the following is grounds for both disqualification and penal action: Creatures in a tournament are not permitted to modify or attack the file system (e.g., create, modify or delete files), the host machine's configuration, user account, network, etc. Creatures may attempt to open network connections, provided that they are not used to attack a system.

Tournament Scoring

Maze tournaments creatures are ranked by their average times for running the same maze **six times**. The lowest time wins.

Natural Selection tournaments run all O(n^4) combinations of four species on the same map. Based on how the game ends, each species receives the following number of points:

Game Ending Condition / Point assignment

Ascension: (on three ascensions by one species)

A	3	First species to complete three ascensions
L	0	Any other species

Domination: (on elimination of ½ of the competitors)

D	3	One of the remaining species
L	0	Eliminated

Exhaustion: (at time limit)

m	2	"Majority": One of the two most-populous species (ties resolved down)
s	1	"Survival": At least one creature alive, but not among the most populous
L	0	Eliminated

The winner of the tournament is the species with the highest total score. Note that keeping a single instance of a creature alive can earn 30% of the total points.

Running the Tournament Software

The Darwin program launched with the “run” script allows you to test your creature against the clock for maze maps or against another creature for natural selection maps. It is very useful for debugging your creature, especially if you use the slow-motion speed and the creature inspector. That is, Darwin is for debugging the **implementation** of your strategy. To test the strategy itself, you want to quickly run many trials with many creatures and see how yours ranks. The best way to do this is to use the tournament software itself.

You can launch the tournament program with a command like:

```
java –cp .:darwin.jar –ea Tournament ns_faceoff Pirate Rover SuperRover Tortoise
```

On Windows, change the colon to a semi-colon. I recommend that you write a little script similar to “run.bat” or “run” that launches the tournament with your creature(s), the ones provided with the SDK, and the ones that your fellow competitors have hopefully shared with you during the training period. That is, I suggest that a good way to develop a creature is to share all of the .class, .png, and .wav files for your creature (but maybe not the .java source), so that everyone can enjoy exploring strategies together and try to explore countermeasures. Note that you should test on lots of different maps (and make some of your own). The tournament will always be run on a map that no-one has seen before.

The SDK creatures provided for you are:

Maze Creatures

- Tortoise
- Skunk
- BamfJG
- PikuLR
- ElephantJL

NS Creatures

- Rover
- SuperRover
- Pirate
- SheepABS
- PsyduckATS
- BrainNRKSLR
- DalekKWDE

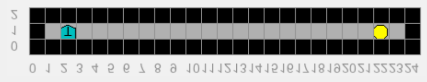
Many of these are inside the darwin.jar file, so you won’t see them in your directory but you can run still against them by listing them on the command line.

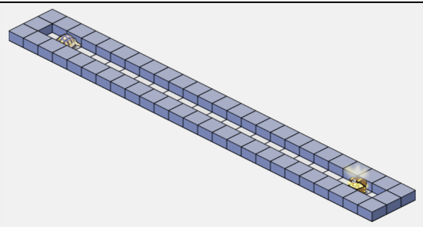
Previous Tournament Winners

Year and League	Player	Creature	Icon
2009 Pro NS *Faceoff*	**Aaron Size**	**SheepABS**	
2011F Williams Maze *Crossroads*	**Josh Geller**	**BamfJG**	
2011F Williams NS *Tanhauser Gate*	**Greg White & Ben Athiwaratkun**	**ProbeGAWPA**	
2011F Pro NS *Tanhauser Gate*	**April T. Shen**	**PsyduckATS**	
2011S Williams Maze *QR*	**Lily Riopelle**	**PikaLR**	
2011S Williams NS *Insurgency*	**Luc Robinson & Nathaniel Kastan**	**BrainNRKSLR**	
2011S Pro NS *Insurgency*	**Kai Wang & Daniel Evangelakos**	**DalekKWDE**	
2012F Williams Maze *2012*	**Jonas Luebbers**	**ElephantJL**	
2012F Williams NS *Metropolis*	**Nigel Munoz**	**BlackMageNW**	
2012F Pro NS *Metropolis*	**Kai Wang & Dan Evangelakos**	**KaledKWDE**	

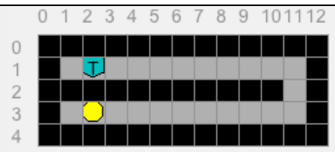
9 / Index of Maps

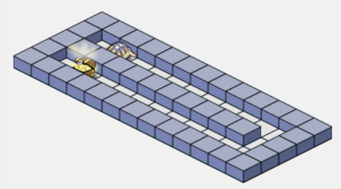
mz_1: “First Try”



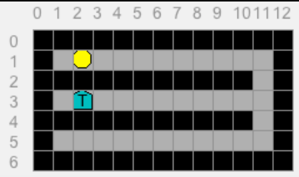


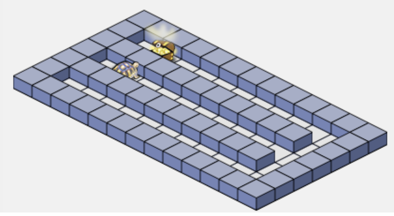
mz_2: “Round the Corner”



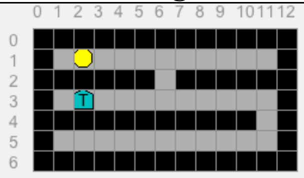


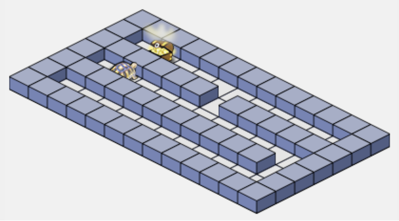
mz_3: “Make a Choice”



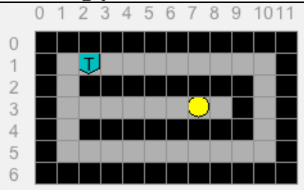


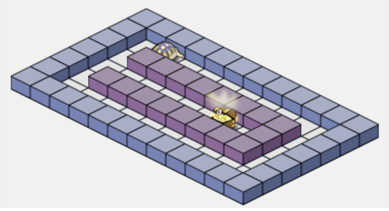
mz_4: “Side Passage”



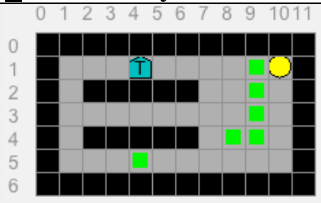
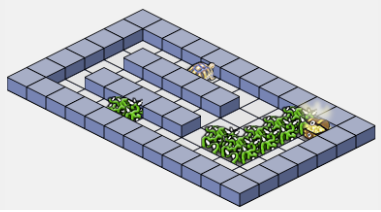


mz_5: “Loopy”

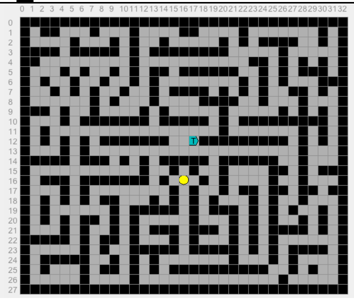
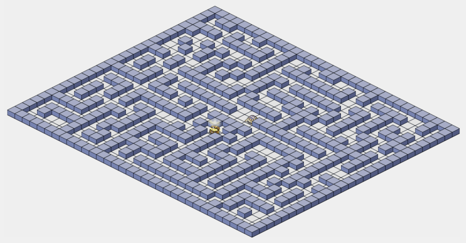




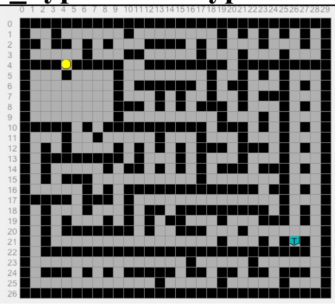
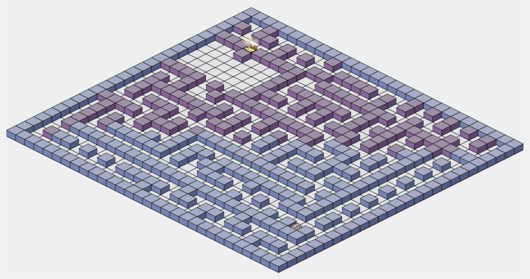
mz_6: “Thorny”

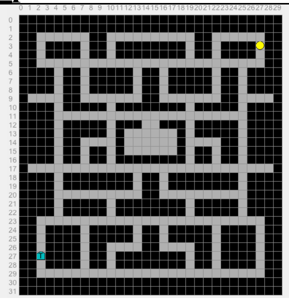
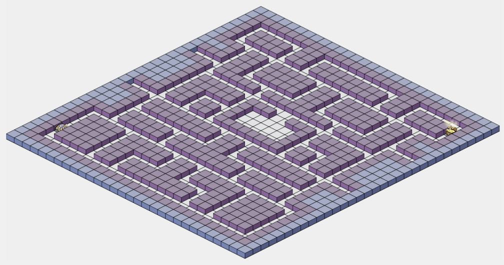
mz_central: “Central”

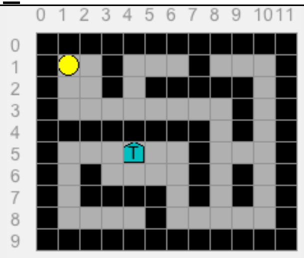
mz_hyperion: “Hyperion Astra”

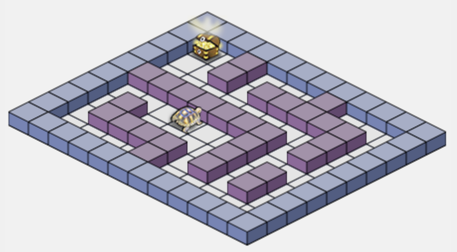



mz_pacman: “Ms. Pac-Man”

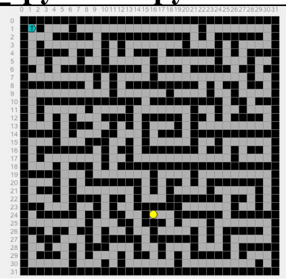



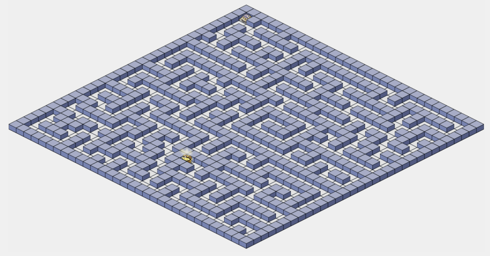
mz_small: “Small”



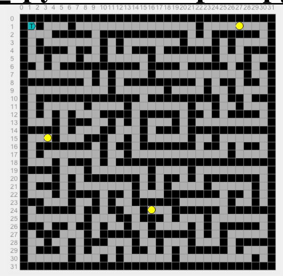


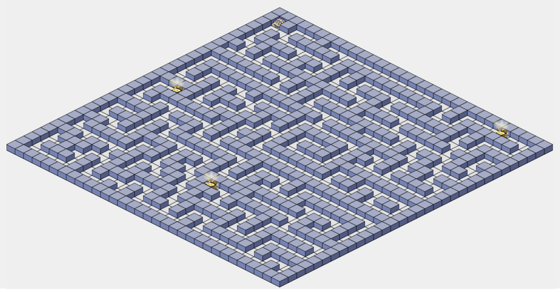
mz_spyrus: “Spyrus”



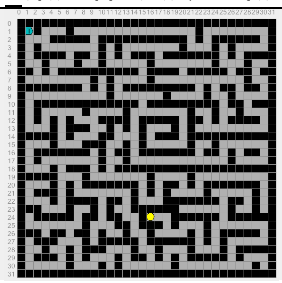
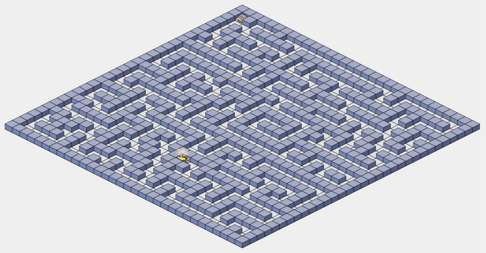


mz_spyrus3: “Triple Spyrus”

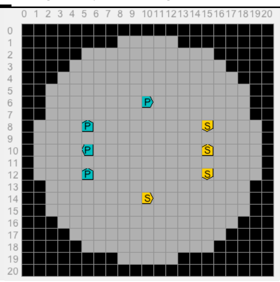
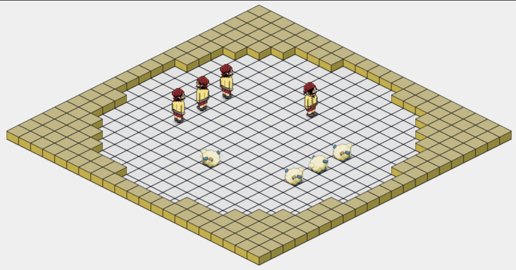




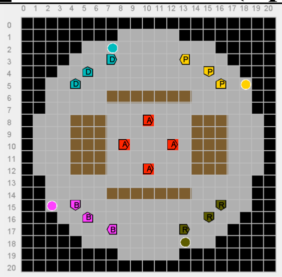
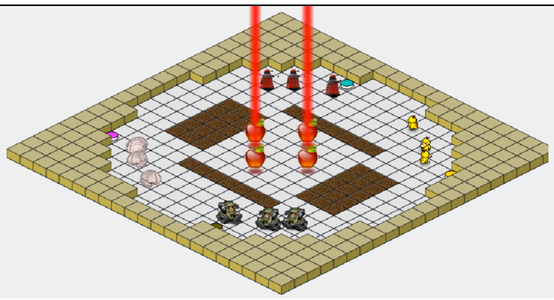
mz_teamcentral: “Team Central”

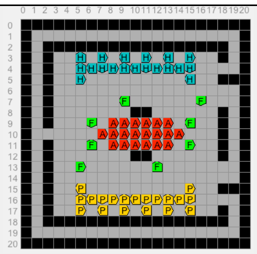
ns_arena: “Arena”

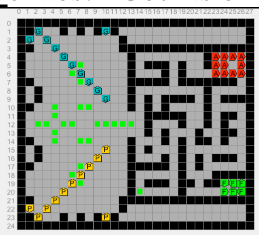
ns_arena4: “Arena (4 player)”

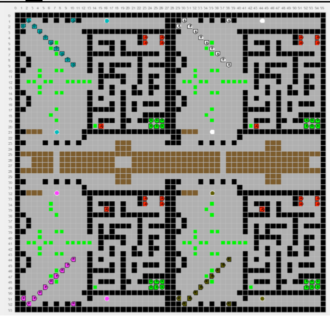
ns_tunnel: “Tunnel”

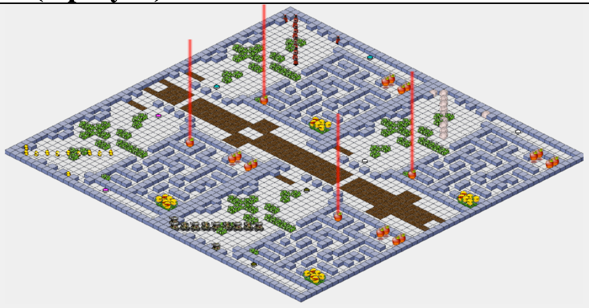


ns_chaos: “Courts of Chaos”

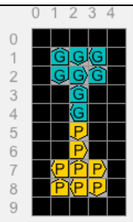


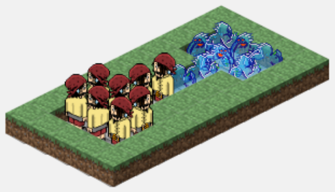
ns_chaos4: “Courts of Chaos (4 player)”



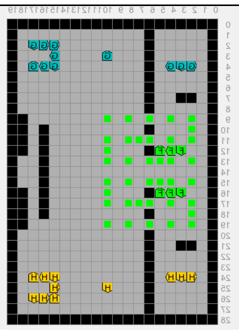


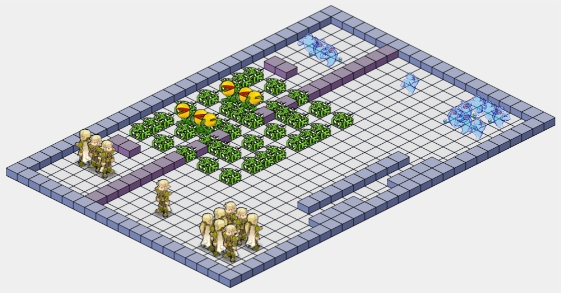
ns_choke: “Chokepoint”



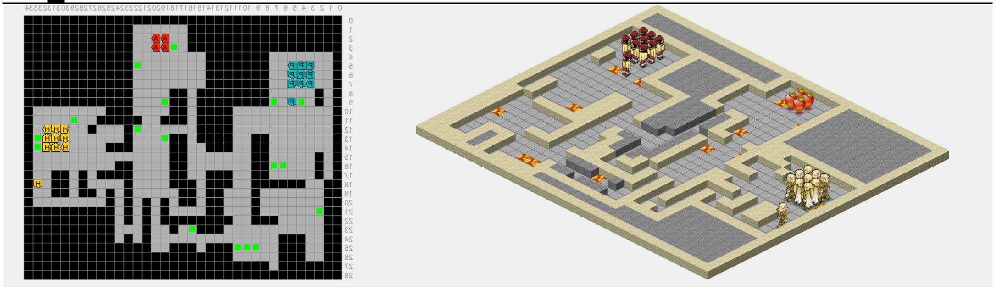


ns_doubletime: “Double Time”

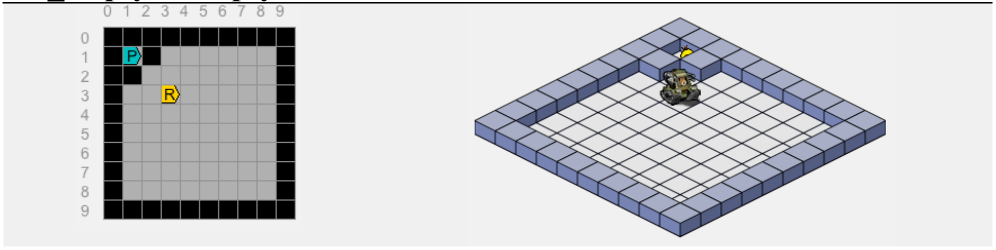




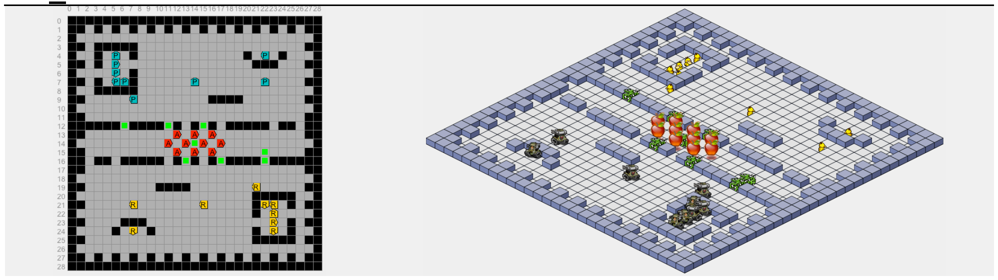
ns_dust: “Dust”



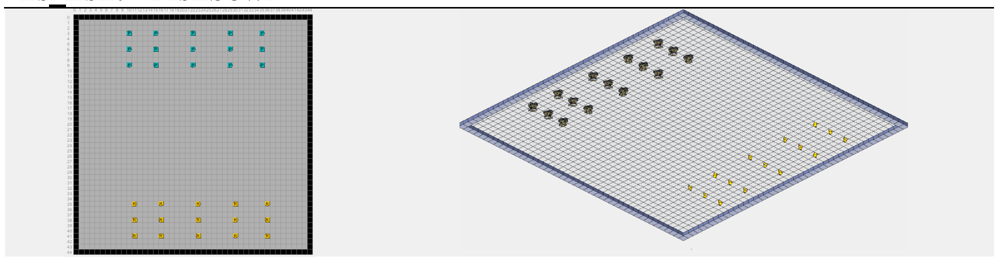
ns_empty: “Empty”



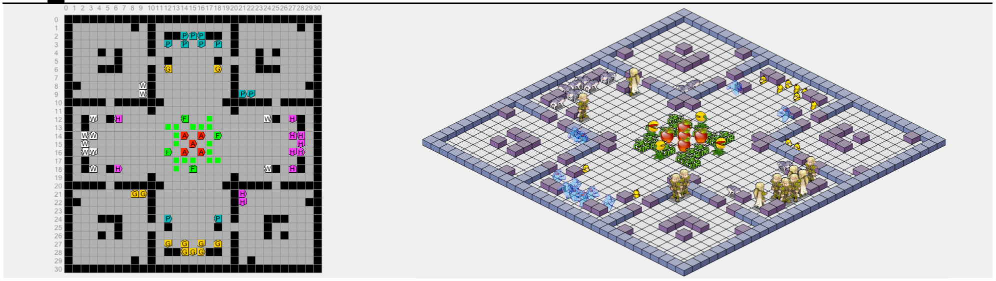
ns_faceoff: “Faceoff!”



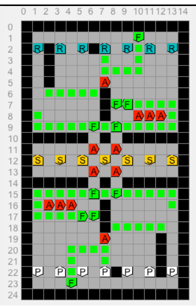
ns_fish: “Fishbowl”



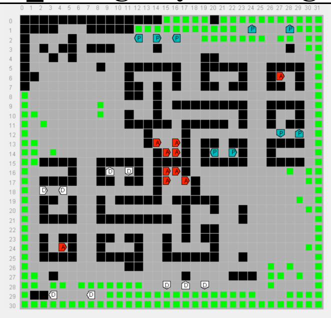
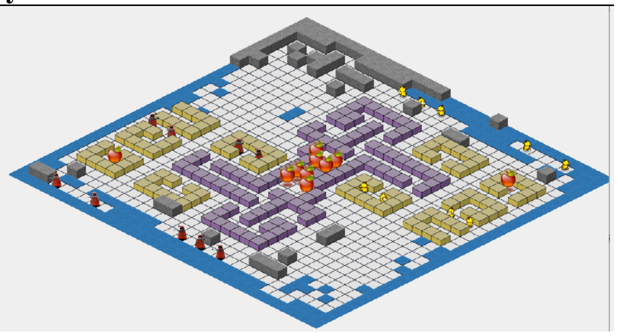
ns_fortress4: “Fortress”



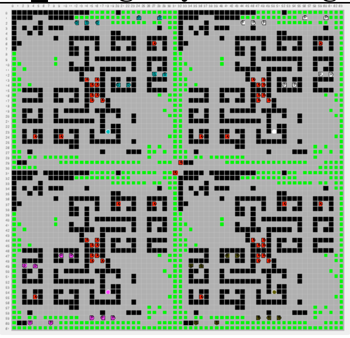
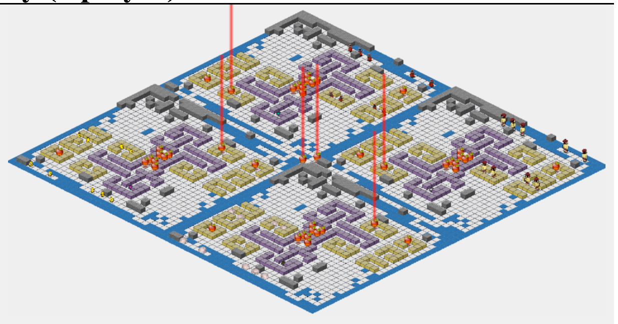
ns_harvest3: “Harvester of Sorrow (3 player)”



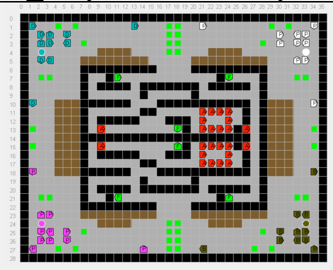
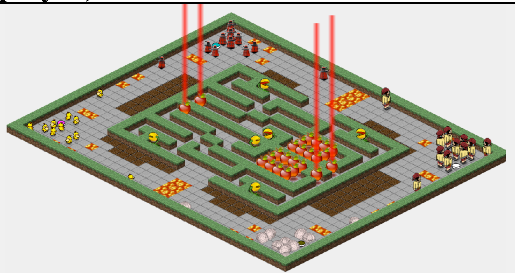
ns_insurgency: “Insurgency”

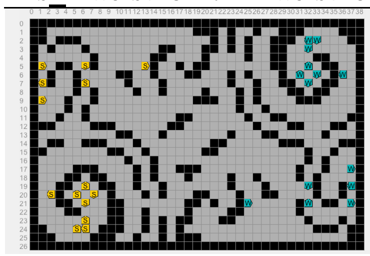
ns_insurgency4: “Insurgency (4 player)”

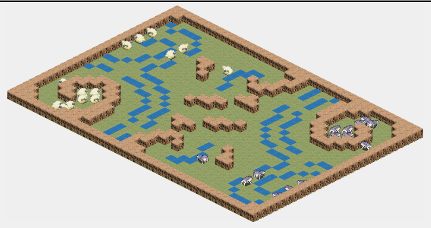



ns_labyrinth4: “Labyrinth (4 player)”

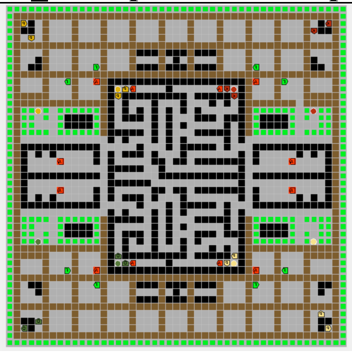



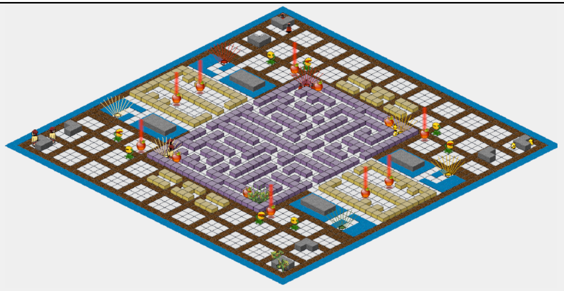
ns_maelstrom: "Maelstrom"



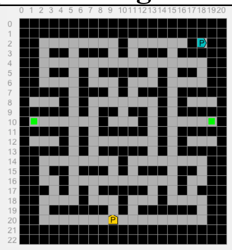


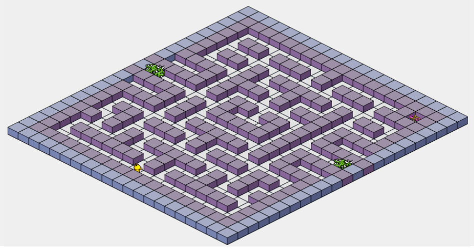
ns_metropolis4: "Metropolis"



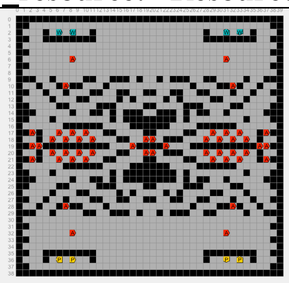


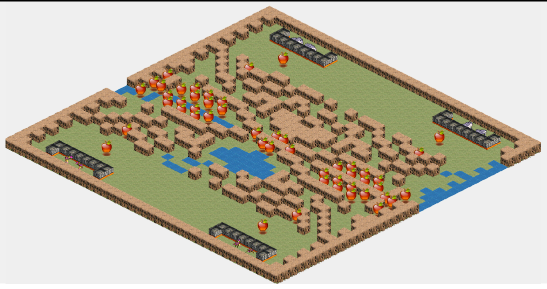
ns_pathfinding: "Pathfinding"



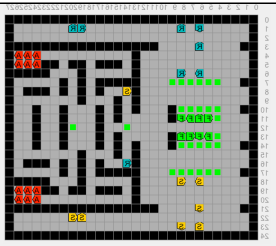


ns_resource: "Resource Race"

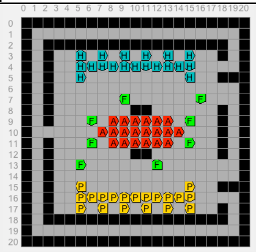




ns_tanhauser: "Tanhauser Gate"



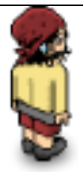
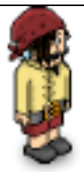
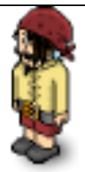
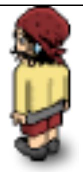
ns_tunnel: “Tunnel”



Advanced Topics

10 / Images

You can customize the way the Simulator renders your Creature in the 3D view by providing four images, called sprites. Each image must be in PNG format and be no larger than 40×60 pixels. The images must be named *Creature*-*D*.png, where *D* is one of N, S, E, W and *Creature* is the name of your creature's class. Below are four images for the Pirate Creature.

			
Pirate-N.png	Pirate-E.png	Pirate-S.png	Pirate-W.png

Images are drawn from a 45-degree isometric perspective. NS and EW lines should be diagonals with a Y:X slope of 1:2. When drawing these it often helps to look at Wall.png to get the perspective right. Since this is the perspective used for many 2D games like Age of Empires, SimCity, Diablo, and Habbo Hotel you can often use sprite images from those games (you can't publicly distribute such sprites, though, because they are copyrighted by the respective developers). A huge list of sprites ripped from 2D games can be found at http://sdb.drshnaps.com/index.htm.

Sprites should have transparent backgrounds. The center of the ground square is in the horizontal center of the sprite and about 8 pixels from the bottom of the sprite. Drawing a subtle drop shadow under a sprite helps makes it appear to actually be standing on the ground.

11 / Sounds

You can customize the sounds that play when your creature takes actions. These must be in .wav audio format and have a limit on their maximum length. The names and length limits are specified below:

Name	**Maximum Length**	**Condition**
Creature-Win.wav	6.5 sec	Victory
Creature-Attack1.wav	1.2 sec	Successful attack
Creature-Attack2.wav	1.2 sec	Successful attack

The attack sounds do not play in the current version of the simulator but are reserved for future use.

12 / Maps

You do not have to create maps to play Darwin, however you may find it useful to make small test maps when debugging your creature. For example, the provided mz_1 through mz_6 maps are simple test cases to help you with basic map navigation.

Maps are ASCII files. The first line optionally begins with a quoted graphics-pack file name. It must then contain the width and height of the map and the map title, separated by spaces and terminated by a newline. The remaining lines form a picture of the map. The elements available are:

- `' '` Empty square
- `'X'`, `'%'` and `'#'` Walls (in different colors)
- `'+'` Hazard
- `'f'` Flytrap (which is a Creature)
- `'a'` Apple (which is a Creature)
- `'e'` Enchanted Apple (which is a Creature)
- `'*'` Treasure (which is a Creature)
- `'0'`…`'9'` Spawn locations of Creature subclasses
- `':’ ,'F','A','E','T'` Empty square, flytrap, apple, enchanted apple, or hazard in fog (experimental, not used in 2012)
- `'.’` Mud
- `'s’` Shrine belonging to the closest spawning species

At load time, the outer border of the map is forced to be all Walls regardless of what was specified in the map file.

By convention, maze maps are named mz_*mapname*.map. They contain a single Treasure (the goal) and a single 0 that is the start position. It is possible to make mazes with multiple treasures; for these all Treasures must be attacked to win. Natural Selection (“deathmatch”) maps are named ns_*mapname*.map and may have any combination of elements.

As an example, the text file for the “Faceoff!” map is shown below.

```
“default.gfx” 29 29 Faceoff!
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX X X X X X X X X X X X X XX
X                           X
XX XXXXX                   XX
X  X 1 X            X 1 X   X
XX X 1               XXX   XX
X    1 X                    X
XX X 11X      1       1    XX
X  XXXXX                    X
XX     1        XXXX       XX
X                           X
XX                         XX
XXXXXX+XXXX+X X+X XXXXX XX XX
XX          a a a          XX
X          a a+a a          X
XX          a a a     +    XX
XX XX XXXXX X+X X+XXXX+XXXXXX
XX                         XX
X                           X
XX        XXXX       0     XX
X                    XXXXX  X
XX     0       0     X00 X XX
X                    X 0    X
XX    XXX              0 X XX
X    X 0 X           X 0 X  X
XX                   XXXXX XX
X                           X
XX X X X X X X X X X X X X XX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

13 / Graphics Packs

You do not have to create graphics packs to play Darwin.

A graphics pack is a file with the extension .gfx and a set of .png images referenced by it. Every map references a graphics pack, or default.gfx by default. The graphics pack changes the appearance of the map in the 3D view. It is purely cosmetic and has no impact on gameplay.

The .gfx file lists the images used for non-Creature map elements as double-quoted strings in the order: wall1 (X), wall2 (#), wall3 (%), thorn (+), floor (‘ ‘), fog (:). All images should be at most 40 pixels wide and drawn on the same isometric 2:1 aspect as characters.

14 / New Features

New in 2012

Mud slows creatures down. This is intended to make pathfinding more interesting.

Enchanted creatures can *ascend* if they step on a *shrine* for their species. Multiple ascensions are a new way to win a game. Creatures spawn enchanted when they are converted from another enchanted creature. When they ascend, they are reincarnated as a non-enchanted creature.

Natural selection maps are about four times larger this year and always feature four competing creatures. This is intended to create more interesting play in the face of creatures that are purely defensive and to encourage collusion and metagaming among players.

Experimental

Fog can exist on any location. Creatures entering a fogged location (even from another fogged location) pay a movement time cost penalty. Fog creates an Observation during observe(), so a creature can only see one square ahead in a fogged area and only creatures on the very edge of a fogged area are visible from outside. I designed this feature to support three strategies in particular: path-finding with variable movement cost, lurking in fog, and hiding in fog. Fog will not appear on any Tournament map this year.