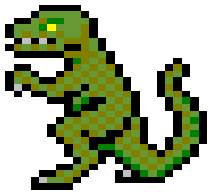
Mash, Mangle & Munch



The Case Study

Mash, Mangle & Munch is a strategy game where the player assumes the roll of a giant monster intent on destroying a sprawling metropolis. There are four working monsters in the game, and the aim of this case study is to build code reading-comprehension by finding a way to add a new monster with different abilities and attributes. Read the game description in "`Mash Mangle Munch.txt`".

The notable files are:

- `MMMDriver.java` - has the main function that runs.

- `MMMPanel.java` - this is the file where you, the student, will enter much of the code for the assignment. The panel has several subclasses that are access utility files. That means that they are not subclasses in a traditional sense: they contain utility methods that have access to the static data fields in `MMMPanel.java`, which feeds access to `AIMovement.java`, `ImageDisplay.java`, `MapBuilder.java`, `Ordinance.java`, `Spawner.java` and `Utilities.java`. This helps keep commonly themed methods in a separate file (like random map building tools can be found in `MapBuilder.java`) and keeps the main panel from being excessively long.

The hierarchy of players, both human controlled and AI are:

Entity.java <− Player.java <− Monster.java <− (Gorilla,java, Dinosar.java, Robot.java, Insect.java)
Entity.java <− Player.java <− Vehicle.java

Other subclass entities are as follows:
Entity.java <− (Bullet.java, Explosion.java, Structure.java)

- `Mash Mangle Munch.txt`- description of the program with keyboard controls and hints.
- `MMM THE ASSIGNMENT.docx` - lists and describes the assignment to be completed (this file).

Your assignment is to add a 5th monster:

Description: The Blop: the offspring of the Bird Flu and the Superbug, the Blop was disposed of by the CDC in a vat of toxic waste. Growing to enormous size with an insatiable appetite, the amorphous mass is out to punish its makers. The Blop leaves a trail of toxic sludge wherever it goes, so it must be careful when making tracks. It can trap victims to eat but lay waste to too much space and the food will never come back. If the Blop is at full health, it can split into two, but at the expense of damaging itself. If it gets close enough to its twin, it can recombine into a single monster to regain health.

Modifications will be made in `MMMPanel.java`. You will want to create a new subclass of Monster, called `Blop.java`. The program runs from `MMMDriver.java`.

1) Create the monster with the following specifications:

name:	"The-Blop"	
animation images:	`playerImages[4]`	
animation delay:	`15`	-sets the speed of the animation frames (same for most monsters)
stomp power:	`20`	-this is how much damage is done to a building when stomped on
speed penalty:	`1`	-affects player speed
reload time:	`0`	-The-Blop only shoots a projectile in EARTH_INVADERS game mode
walk damage:	`90`	-this is how much damage is done just by walking on a structure
projectile type:	"`SLUDGE`"	–only used for EARTH_INVADERS game mode for The-Blop
burn damage:	`50`	-this is how much damage the monster takes by standing in a fire
ABILITIES:		
swimmer:	`yes`	-we move faster in water
can split:	`yes`	-we can split into an AI controlled twin at the expense of health
impervious to bullets:	`yes`	-bullets do no damage, but shells and flame do double damage
slime trail:	`yes`	-we leave a trail of toxic slime behind us (Blop-Glop)
can eat all:	`yes`	-we can grab and eat any type of unit

Note: monsters are created in several places in `MMMPanel.java` (for player1, player2, AI player, etc.) as well as in `Spawner.java` (which spawns AI monsters in the game mode `CITY_SAVERS`). To make the game activate when you select the Blop, search for the variable `needToStart` in `MMMPanel`.

2) The-Blop can grab any type of unit to eat.

3) The-Blop gains 5-10 health points for any type of unit that is eaten, and its hunger level decreases by one level.

4) Complete the code in `MMMPanel` to add the functionality for creatures that are impervious to bullets:

4a. Any monster that is impervious to bullets takes no damage from bullets (Projectile of type "`BULLET`") but takes double damage from all others. Note: this rule should NOT be applied in `EARTH_INVADERS` mode and should be applied to any kind of monster that has this feature. Press 'Y' in-game to increase the threat level to test your code.

4b. If Projectile of type "`BULLET`" hits a creature that is impervious to bullets, there should not be an explosion image produced (the bullet is absorbed).

5) For any ground unit that The-Blop runs over, it should produce a smoke puff explosion instead of a fireball explosion. Exclude this rule for trains, boats, and aircraft.

6) For any creature with the slime trail ability, with each space that the monster moves, it leaves a trail of toxic slime called "`sludge`". Blop-glop cannot be dropped in the water or over another structure (even rubble). A pile of sludge takes the form of a `Structure` (like buildings, trees or rubble). This way, vehicles and crowds should not be able to pass over it.

`board` is the array of terrain, `structures` is the array of `Structure` objects.
A terrain element in `board` that contains a "`~`" (tilde) has water.

Sludge does property damage by destroying roads and parks, between $0-$2999 damage.

Sludge does not need to be a subclass of a `Structure`, but an instance of a `Structure` object.

name: "`sludge`"
images: `sludgeImages`
animationDelay: `15`
is passable: `true` (passable by monsters/tanks/aircraft, impassable by everyone else)
is destroyable: `true` (not destroyable by stomp, but should be destroyed by tank shells or fire)
height: `1`
health: `-1` (-1 health means it can be shot over by projectiles)
image index: random `0-3`: pick a random one from blopGlopImages
property value: `0`

For leaving the trail of sludge, investigate `Structure.java` to see how the constructor of Structures are defined, and `MapBuilder.java` for insight as to how to create instances of Structures. You will also find in `MapBuilder.java` the means in which water is defined in the world, so that you can keep sludge from being dropped over water.

7) The Blop can't shoot projectiles (outside of the `EARTH_INVADERS` game mode), but if the player tries to hit the shoot key (or mouse button), make the Blop separate into two entities, the original player-controlled monster, and an AI controlled twin. The Blop can only separate if its health is 100. When it separates, the original Blop should lose a random number of health points between 25 and 75. The Blop cannot separate again if its twin is active, even if its health gets back up to 100. If the Blop is close to its twin (within half of a city block), hitting the shoot key will recombine them into a single monster. When the Blop recombines, it gains between 50 and 100 health points.

Note: this rule should not be applied for `EARTH_INVADERS` game mode. There is a boolean data-field called `hasSplit` that can be used to flag whether it has split yet. There is also a static int index to mark the spot where the twin goes in the players array called `SPLIT_TWIN`. Look in `MMMPanel` for how AI friends are added and removed for inspiration on how you can add a twin and remove it when you recombine. The method that will hold the splitting logic is in `MMMPanel` and is called:

```
public static void splitTwin(Monster curr)
```

8) The `highScores` array is read from (and writes into) a text file for each game mode. These files are vulnerable to shenanigans: someone could just open it up in a text editor and insert their fake high score. Create a means of encrypting the data when it is written into the file and decrypt it when it is read from the file. If a user were to open it with a text editor, they should not be able to tell that it is the high score data. But from the scope of the program, it should work the same.

Note: delete all the old high-score text files so that they can be rebuilt as encrypted. You can find shells for an encrypt and decrypt methods in `Utilities.java`. As additional protection, you can rename the scores folder and the names of the files to something less obvious and tempting to click on.

```
KEYBOARD CONTROLS (tap keys - do not hold down):

movement:          arrow keys (up, left, down, right), or W,A,S,D.
                   Or move mouse relative to monster positon.
                   Put the mouse in motion for the monster to move.
shoot/throw:       Enter or mouse left click.
eat:               Shift
grab:              Ctrl
jump/stomp:        Space or mouse right click.
turn head:         <, >, or use the center mouse wheel
shoot air targets:Space + Enter

quit:              Esc to back out to the main menu or quit from main menu.
pause:             p
toggle map/keys:   m
change game speed :     Keypad [,]
change screen size:     Keypad +, -
```

Extra credit: create a 6th monster type - Wormoid.
DESCRIPTION:
Wormoid: while fracking for natural gas, the Shadow Government Energy Conglomerate (SGEC) disturbed the subteranian nest of Wormoid. Agitated by the noise pollution from SGEC's drilling and pounding, Wormoid is out to silence her noisy neighbors upstairs. Wormoid can not jump, but she sure can dig: rendering her invisible to unsuspecting civilians or the pesky military machines. But be careful not to dig under electrical towers, gas stations or water: Wormoid can't swim, and is vulnerable to electricity and fire. Wormoid has no arms, but she can spit a sticky web to trap food for later. Her web has another nifty function: Chain a web between two close towers to create a trap for low-flying aircraft.

1) Create the monster with the following specifications in `Worm.java`.

name:	`"Wormoid"`	
animation images:	`playerImages[5]` with a delay of animation_delay/2	
stomp power:	`30`	-this is how much damage is done to a building when stomped on
speed penalty:	`0`	-affects player speed
reload time:	`100`	-amount of time to recharge a web
walk damage:	`5`	-this is how much damage is done just by walking on a structure
projectile type:	`"WEB"`	-traps ground units or sets a trap between tall towers
burn damage:	`50`	-this is how much damage the monster takes by standing in a fire
ABILITIES:		
digger:	`yes`	-we can dig underground
shooter:	`yes`	-we can shoot a projectile
can't grab:	`yes`	-we have no arms to grab and hold food with

NOTE: monsters are created in several places `in MMMPanel.java` (for player1, player2, AI player, etc) as well as in `Spawner.java` (which spawns AI monsters in the game mode `CITY_SAVERS`).

2) The Wormoid has no arms, so she cannot grab and hold on to any ground units. This is true for any monster with the `cantGrab` feature. When this kind of monster initiates a grab attempt, she should just eat the unit within snatching range:

* Wormoid can grab all units except aircraft (`AIR`) and boats (`BOAT`).
* Eating any type of unit will reset her hunger level back to zero.
* Eating a unit of type `CROWD` restores 5 health points.
* Eating a bus will restore 4 health points.
* Eating any other unit will restore 2 health points.

3) Wormoid should only be able to shoot a web if she has met her reload time and is not weakened from hunger.

4) When any digger initiates a stomp command, it digs a hole and goes underground (or resurfaces if already underground). Both digging and resurfacing should leave a "hole" structure on that location. Digging holes is not allowed in `EARTH_INVADERS` or `BOMBER_`DODGER game modes.

This will define an instance of a `Structure` object:
name: "`hole`"
images: `holeImages`
is passable: `true` (passable by monsters/aircraft, impassable by everyone else)
is destroyable: `false`
height: `1`
health: `-1` (-1 health means it can be shot over by projectiles)
image size: `cellSize/3`
image index: pick a random one from `holeImages`
property value: `0`

For leaving the hole, look into `Structure.java` to see how the constructor of Structures are defined, and `MapBuilder.java` for insight as to how to create instances of Structures. You will also find in `MapBuilder.java` the means in which water is defined in the world, so that you can keep holes from being dropped over water.

Also look for where the code allows for the WoeMantis to fly when the stomp command is issued. You will find this in `MMMPanel` and `Utilites.java`.

Any human, vehicle, or player 2 spawn points at the location of the hole should be removed from the list of spawn points.

$3000 additional property damage should be dealt for the hole left in the ground.

4b) There is no collision between any vehicles and a digger if it is underground.

4c) Diggers can not shoot a web while underground.

4d) While underground, diggers are invisible except for a small smoke puff that appears any time the digger initiates or completes a move command. Use a water explosion if digging under water and an electrical explosion if under an electric tower.

4e) While underground, the digger should be able to traverse to a location above or below it even if there is a structure there.

4f) Diggers only moves at 1/3 speed when underground.

5) A digger surfacing from under an electrical tower or water causes instant death. Look to `MapBuilder.java` to see how board locations store strings to denote electrical towers and water.

6) A digger initiating a stomp to dig or surface should cause a large explosion (the same as the one caused by a WoeMantis that is landing). The resulting radius damage should do 50 points of damage with a 5% chance of causing a fire.

7) Every 100 frames, a digger will take a random amount of damage when under water:
25-75 points of damage when in water but digging underneath it, and 50-150 points of damage if in the water but not digging underneath it. Check in `Utilities.java` for damage done over time due to drowning.

8) Every 100 frames, a digger will take 25-75 points of damage if digging under an electric tower.

9) Diggers takes half damage from any radius damage when digging underground.
Check `Ordinance.java` for radius damage.

10) AI units should not be able to see diggers when digging underground.
Check `AImovement.java` for AI vision of monsters.

11) If a unit is struck by a web (excluding trains, other monsters, and air vehicles), set the stun time to 4 times the `messageTime`.

12) Any player struck by a web should produce a small smoke puff explosion

13) Any monster with the web projectile can set traps for high flying aircraft by stringing a web between two towers that are within 3 square cellSize blocks of one another. Currently, code exists in `Ordinance.java`'s **`outOfBounds`** method that records information about each web trap that is set (for use in displaying the trap and collision checking). There is an array of ArrayLists (one ArrayList for each of the 9 panels) called `webs`. Each element of each List contains an array of 8 integers: the first two values record the x & y position of the first web node, followed by the x & y position of the second web node. The last 4 values record the row and columns for the two buildings that are the supports for the web trap. For example, lets say the center panel has a webs List with the following values:
[124, 43, 49, 213, 4, 5, 7, 6]. That means that there is one web trap between (124, 43) and (49, 213) in pixel space, chained between buildings at (4,5) and (7,6) in row/column space. In `AImovement.java`'s `makeEnemyMove` method, this information is used to see if there is a collision between an aircraft and a web trap that calls `Utilities.isPointOnSegment`.

```
//pre:  x1 - x2 != 0
//post: returns true if the point (x,y) is within (cellSize/2) units from the
//      line segment (x1,y1) & (x2, y2)
boolean isPointOnSegment(int x, int y, int x1, int y1, int x2, int y2)
```

Consider this: given point (x,y) and a line segment defined by (x1,y1) and (x2,y2), how could you determine if the point is on the segment? How could you determine if the point is close to the segment? Draw a few pictures and note that there is a useful helper method defined right above `isPointOnSegment` in `Utilities.java`.