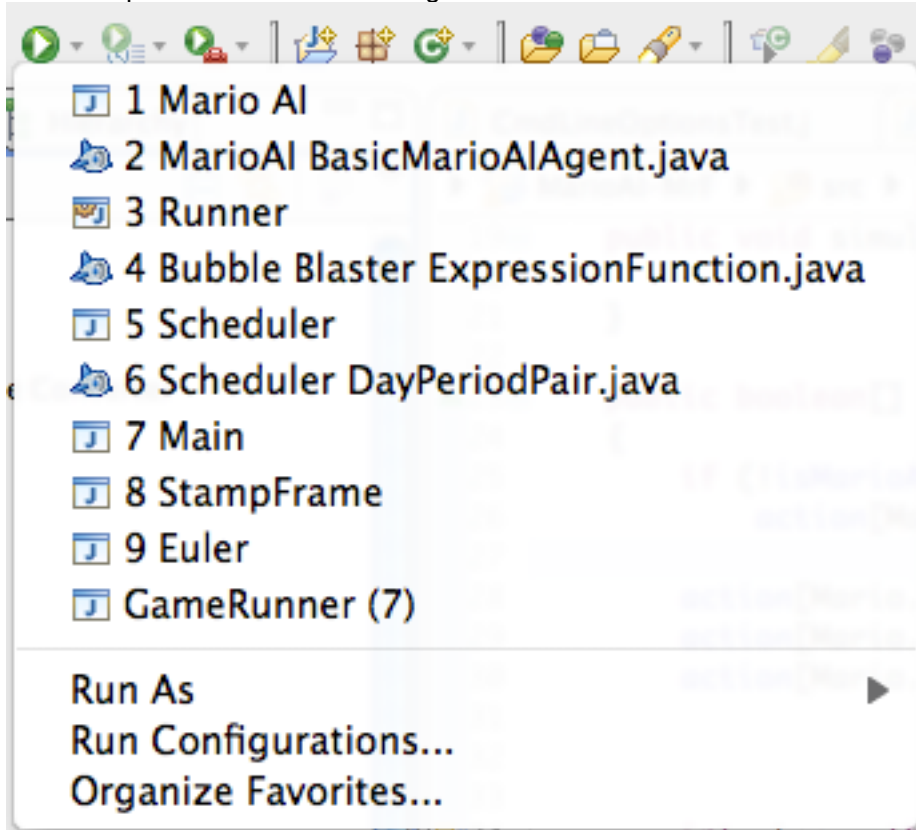


Set Up

- 1) Download the Mario project data from my website - DO NOT EXTRACT
- 2) In Eclipse, go to **File->Import...** Under **General**, select “**Existing Projects into Workspace**”. Click **Next**
- 3) Choose the second option: “**Select archive file**”. Click the **Browse** button and find the file you downloaded. **Finish** the import.

Running

- 4) Click on the drop down arrow next to the green launch button in the toolbar.



- 2) Select “**Run Configurations...**”
- 3) Click on the icon in the upper left hand corner with a plus symbol to create a new launch configuration.
In the Main tab:
 - a) Name: Mario AI
 - b) Project: Mario AI
 - c) Main-Class: ch.idsia.scenarios.Main
In the Arguments tab:
 - a) Program Arguments: `-ag ch.idsia.agents.controllers.ForwardJumpingAgent -ld 0`
 - in the future, change ForwardAgent to the name of your Mario AI
 - change the 0 to a bigger number to pick a harder level
- 4) Click Apply and then Close
- 5) Now when you run the simulation, the ForwardAgent should take off. ‘w’ will make the game end immediately

Creating an AI

1) In the `ch.idsia.agents.controllers` package, make a subclass of `BasicMarioAgent`. It should look something like this:

```
package ch.idsia.agents.controllers;

import ch.idsia.benchmark.mario.environments.Environment;

public class SecretAgent extends BasicMarioAIAgent {

    public SecretAgent() {
        super("My Mario AI");
        reset();
    }

    public boolean[] getAction()
    {
        //action[Mario.KEY_LEFT] = true/false
        //action[Mario.KEY_RIGHT] = true/false
        //action[Mario.KKEY_DOWN] = true/false
        //action[Mario.KEY_JUMP] = true/false
        //action[Mario.KEY_SPEED] = true/false

        return action;
    }

    public void reset()
    {
        action = new boolean[Environment.numberOfButtons];
    }
}
```

2) I'm overriding the `getAction()` method that is inherited from `BasicMarioAgent`. In this method, you decide which buttons to have pressed. The action array will remember what buttons you were pressing in the previous time step. By toggling the values between true and false, you can get Mario to perform any operation he would in a regular game.

3) There are several fields that you should know that you inherit from your subclass (and have direct access to - but cheating doesn't work... you can change their values, but it won't mess up the simulation)

- `isMarioOnGround` (boolean)
- `isMarioAbleToJump` (boolean)
- `isMarioAbleToShoot` (boolean)
- `isMarioCarrying` (boolean)
- `marioStatus` (int): `Mario.STATUS_RUNNING`, `Mario.STATUS_WIN`, or `Mario.STATUS_DEAD`
- `marioMode` (int): 0 == small, 1 == big, 2 == fireball

4) Jump example - to get Mario to jump, you can't just hold down the jump button like so:

```
action[Mario.KEY_JUMP] = true;
```

You actually have to release the key at some point and repress it. Here's a standard example:

```
if (!isMarioAbleToJump && isMarioOnGround)
    action[Mario.KEY_JUMP] = false;
else
    action[MARIO.KEY_JUMP] = true;
```

This will allow Mario to jump up and down continuously.

5) The last thing you need to know is about the environment around Mario.

The perceived enemies and level:

- receptiveFieldWidth (int): how far Mario can see left and right in blocks
- receptiveFieldHeight (int): how far Mario can see up and down in blocks
- marioCenter (int[]): an array of length 2 having the row and column that Mario is at inside of his receptive vision
- levelScene (byte[][]): 2D array specifying where Mario is in relation to everything else (Mario will be at the center, so levelScene[marioCenter[0]][marioCenter[1]] is the object Mario is currently interacting with - it's usually the same as levelScene[receptiveFieldHeight/2][receptiveFieldWidth/2])
- Values in the levelScene 2D array can include the following (you can't use the fields... they are private... for some reason)

```
CANNON_MUZZLE = -82;
CANNON_TRUNK = -80;
COIN_ANIM = Sprite.KIND_COIN_ANIM; //1
BREAKABLE_BRICK = -20;
UNBREAKABLE_BRICK = -22; //a rock with animated question mark
BRICK = -24; //a rock with animated question mark
FLOWER_POT = -90;
BORDER_CANNOT_PASS_THROUGH = -60;
BORDER_HILL = -62;
FLOWER_POT_OR_CANNON = -85;
NOTHING = 0
```

- enemies (byte[][]): 2D array specifying where Mario is in relation to enemies on the screen (Mario will be at the center, so enemies[marioCenter[0]][marioCenter[1]] is the object Mario is currently interacting with)
- Values in the enemies 2D array can be found at the top of the Sprite class. You can access them directly via Sprite.*** (oddly, these are public). I'm not sure if Mushrooms and the like end up in the enemies 2D array or the level scene 2D array. Some values include:

```
KIND_NONE = 0;
KIND_MARIO = -31;
KIND_GOOMBA = 80;
KIND_GOOMBA_WINGED = 95;
KIND_RED_KOOPA = 82;
KIND_RED_KOOPA_WINGED = 97;
KIND_GREEN_KOOPA = 81;
KIND_GREEN_KOOPA_WINGED = 96;
KIND_BULLET_BILL = 84;
KIND_SPIKY = 93;
KIND_SPIKY_WINGED = 99;
KIND_ENEMY_FLOWER = 91;
```

```
KIND_SHELL = 13;  
KIND_MUSHROOM = 2;  
KIND_FIRE_FLOWER = 3;  
KIND_PARTICLE = 21;  
KIND_SPARCLE = 22;  
KIND_COIN_ANIM = 1;  
KIND_FIREBALL = 25;  
KIND_UNDEF = -42;
```

- mergedObservations (byte[][]): 2D array containing all the enemy and level information in 1 array

- marioFloatPos and enemiesFloatPos (float[]): I've never used this, but I think it gives mario and all enemy positions in the world (not just in Mario's perception... but I'm not sure)