**import** java**.**awt**.**image**.\*;**

**import** java**.**io**.\*;**

**import** java**.**util**.\*;**

**import** javax**.**imageio**.\*;**

**import** java**.**awt**.**Point**;**

**import** org**.**w2mind**.**net**.\*;**

**import** java**.**io**.**Serializable**;**

public class PacManDBZworld **extends** AbstractWorld

**{**

//this will be 16\*16

//yields 256 squares

public static final int GRID\_SIZE **=** 16**;**

protected Vector**<**Point**>** freeCells**;**

protected Vector**<**Point**>** walledCells **=** **new** Vector**<**Point**>();**

Point friezaPos**,** gokuPos**,** dbPos**,** ginyuPos**,**cellPos**;**// positions of avatars

int MAX\_STEPS **=** 50**;** // number of steps in a run

int numCaught**;** // primary score field = number of times robber has been caught in this run

List **<**String**>** scoreCols**;** // Headers for the score fields

//an int to say if we collected 3 dragonballs

//this will be passed to the mind

int dbCollected **=** 0**;**

//a count of number dragonballs collected

//collect more than enemies kill you to win

int dbCount**;**

int timestep**;**

boolean friezaDied **=** **false;**

boolean ginyuDied **=** **false;**

boolean cellDied **=** **false;**

boolean run1 **=** **true;**

protected int gokuHealth**;**

protected int friezaHealth**;**

protected int ginyuHealth**;**

protected int cellHealth**;**

//when this is 0

//game ends

int numBadGuys **=** 3**;**

// actions (public static since used by Mind):

// custom for 2d

public static final int ACTION\_LEFT **=** 0**;**

public static final int ACTION\_RIGHT **=** 1**;**

public static final int ACTION\_UP **=** 2**;**

public static final int ACTION\_DOWN **=** 3**;**

public static final int STAY\_STILL **=** 4**;**

public static final int NO\_ACTIONS **=** 5**;**

public static final int ACTION\_RAND **=** 6**;**

//images used

String SUPPORT\_DIR **=** "images"**;**// support files

String IMG\_FRIEZA **=** SUPPORT\_DIR **+** "/frieza.png"**;**

String IMG\_GOKU **=** SUPPORT\_DIR **+** "/gokuSmall.png"**;**

String IMG\_CLOUD **=** SUPPORT\_DIR **+** "/cloud2.png"**;**

String IMG\_BG **=** SUPPORT\_DIR **+** "/bg.jpg"**;**

String IMG\_DRAGONBALL **=** SUPPORT\_DIR **+** "/dragonball.png"**;**

String IMG\_DEAD **=** SUPPORT\_DIR **+** "/dead.png"**;**

String IMG\_GINYU **=** SUPPORT\_DIR **+** "/ginyu.png"**;**

String IMG\_SUPER **=** SUPPORT\_DIR **+** "/super.png"**;**

String IMG\_LIGHT **=** SUPPORT\_DIR **+** "/light.png"**;**

String IMG\_CELL **=** SUPPORT\_DIR **+** "/cell.png"**;**

// transient - don't serialise these:

private transient ArrayList **<**BufferedImage**>** buf**;**

private transient InputStream friezaStream **=** **null,** gokuStream **=** **null,** cloudStream **=** **null,** bgStream **=** **null,** dragonballStream **=** **null,**

deadStream **=** **null,** ginyuStream **=** **null,**superStream **=** **null,**lightStream **=** **null,** cellStream **=** **null;**

private transient BufferedImage friezaImg**,** gokuImg**,** cloudImg**,**bgImg**,**dragonballImg**,**deadImg**,** ginyuImg**,**superImg**,**lightImg**,**cellImg**;**

//stores images width + height

int imgwidth**,** imgheight**;**

//---------------------------------------

//Return a random valid position on the grid

//--------------------------------------

protected Point randomPosition**()**

**{**

Random r **=** **new** Random**();**

Point a **=** **new** Point**(**r**.**nextInt**(**GRID\_SIZE**),**r**.**nextInt**(**GRID\_SIZE**));**

**while(!**validPosition**(**a**))**

**{**

a **=** **new** Point**(**r**.**nextInt**(**GRID\_SIZE**),**r**.**nextInt**(**GRID\_SIZE**));**

**}**

**return** a**;**

**}**

//-----------------------------------------

//Give all characters valid random board positions

//-----------------------------------------

private void initPos**()**

**{**

friezaPos **=** randomPosition**();**

**do**

**{**

gokuPos **=** randomPosition**();**

ginyuPos **=** randomPosition**();**

cellPos **=** randomPosition**();**

**}**

// repeat until different position

**while(**friezaPos**.**equals**(**gokuPos**)** **&&** ginyuPos**.**equals**(**gokuPos**)** **&&**

ginyuPos**.**equals**(**friezaPos**)** **&&** cellPos**.**equals**(**gokuPos**)** **&&** cellPos**.**equals**(**friezaPos**)** **&&** cellPos**.**equals**(**ginyuPos**));**

**}**

//----------------------------------------------

//return with a random move from 0 - (NO\_ACTIONS -1)

//----------------------------------------------

private int randomAction**()**

**{**

Random r **=** **new** Random**();**

**return** **(**r**.**nextInt**(**NO\_ACTIONS **-** 1**));**

**}**

//Check is is ok to move left

public boolean validLeft**(**Point startPos**)**

**{**

Point backUp **=** **new** Point**(**startPos**.**x**,**startPos**.**y**);**

backUp**.**x **=** **(**backUp**.**x **-** 1**);**

**if(**validPosition**(**backUp**)** **==** **false)**

**{**

**return** **false;**

**}**

**return** **true;**

**}**

//check is it ok to move right

public boolean validRight**(**Point startPos**)**

**{**

Point backUp **=** **new** Point**(**startPos**.**x**,**startPos**.**y**);**

backUp**.**x **=** **(**backUp**.**x **+** 1**);**

**if(**validPosition**(**backUp**)** **==** **false)**

**{**

**return** **false;**

**}**

**return** **true;**

**}**

public boolean validUp**(**Point startPos**)**

**{**

Point backUp **=** **new** Point**(**startPos**.**x**,**startPos**.**y**);**

backUp**.**y **=** **(**backUp**.**y **-** 1**);**

**if(**validPosition**(**backUp**)** **==** **false)**

**{**

**return** **false;**

**}**

**return** **true;**

**}**

public boolean validDown**(**Point startPos**)**

**{**

Point backUp **=** **new** Point**(**startPos**.**x**,**startPos**.**y**);**

backUp**.**y **=** **(**backUp**.**y **+** 1**);**

**if(**validPosition**(**backUp**)** **==** **false)**

**{**

**return** **false;**

**}**

**return** **true;**

**}**

// Move in direction supplied

private void move**(**Point startPos**,** int direction**)**

**{**

**if(**direction **==** ACTION\_LEFT**)**

**{**

**if(**validLeft**(**startPos**))**

**{**

startPos**.**x**=((**startPos**.**x **-** 1 **+** GRID\_SIZE**)** **%** GRID\_SIZE**);**

**}**

//If avatar gets stuck use a random move to try and free him

**else**

**{**

int i **=** randomAction**();**

move**(**startPos**,** i**);**

**}**

**}**

**if(**direction **==** ACTION\_RIGHT**)**

**{**

**if(**validRight**(**startPos**))**

**{**

startPos**.**x**=((**startPos**.**x **+** 1 **+** GRID\_SIZE**)** **%** GRID\_SIZE**);**

**}**

**else**

**{**

int i **=** randomAction**();**

move**(**startPos**,** i**);**

**}**

**}**

**if(**direction **==** ACTION\_UP**)**

**{**

**if(**validUp**(**startPos**))**

**{**

startPos**.**y**=(**startPos**.**y **-** 1**);**

**}**

**else**

**{**

int i **=** randomAction**();**

move**(**startPos**,** i**);**

**}**

**}**

**if(**direction **==** ACTION\_DOWN**)**

**{**

**if(**validDown**(**startPos**))**

**{**

startPos**.**y**=(**startPos**.**y **+** 1**);**

**}**

**else**

**{**

int i **=** randomAction**();**

move**(**startPos**,** i**);**

**}**

**}**

**if(**direction **==** ACTION\_RAND**)**

**{**

int i**;**

i **=** randomAction**();**

**if(**i **==** ACTION\_LEFT**)**

**{**

**if(**validLeft**(**startPos**))**

**{**

startPos**.**x**=((**startPos**.**x **-** 1 **+** GRID\_SIZE**)** **%** GRID\_SIZE**);**

**}**

**else**

**{**

**do**

**{**

i **=** randomAction**();**

**}**

**while(**i **==** ACTION\_LEFT**);**

move**(**startPos**,** i**);**

**}**

**}**

**if(**i **==** ACTION\_RIGHT**)**

**{**

**if(**validRight**(**startPos**))**

**{**

startPos**.**x**=((**startPos**.**x **+** 1 **+** GRID\_SIZE**)** **%** GRID\_SIZE**);**

**}**

**else**

**{**

**do**

**{**

i **=** randomAction**();**

**}**

**while(**i **==** ACTION\_RIGHT**);**

move**(**startPos**,** i**);**

**}**

**}**

**if(**i **==** ACTION\_UP**)**

**{**

**if(**validUp**(**startPos**))**

**{**

startPos**.**y**=(**startPos**.**y **-** 1**);**

**}**

**else**

**{**

**do**

**{**

i **=** randomAction**();**

**}**

**while(**i **==** ACTION\_UP**);**

move**(**startPos**,** i**);**

**}**

**}**

**if(**i **==** ACTION\_DOWN**)**

**{**

**if(**validDown**(**startPos**))**

**{**

startPos**.**y**=(**startPos**.**y **+** 1**);**

**}**

**else**

**{**

**do**

**{**

i **=** randomAction**();**

**}**

**while(**i **==** ACTION\_DOWN**);**

move**(**startPos**,** i**);**

**}**

**}**

**}**

**}**

//-------------------------------

//check if the game has ended

//-------------------------------

private boolean runFinished**()**

**{**

**return** **(** timestep **>=** MAX\_STEPS **);**

**}**

//==========================================================

//add the points where walls will be, add to vector

//==========================================================

public void genWalls**()**

**{**

//-------------------------------

//Outer walls

//left wall

**for(**int i **=** 0**;**i **<** GRID\_SIZE**;**i**++)**

**{**

//leaving opening where appropriate

**if(**i **==** 7 **||** i **==** 8**)**

**{**

//do nothing

**}**

**else**

**{**

Point p **=** **new** Point**(**0**,**i**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

//top wall

**for(**int i **=** 0**;**i **<** GRID\_SIZE**;**i**++)**

**{**

Point p **=** **new** Point**(**i**,**0**);**

walledCells**.**addElement**(**p**);**

**}**

//right wall

**for(**int i **=** 0**;**i **<** GRID\_SIZE**;**i**++)**

**{**

//leave appropriate openings

**if(**i **==** 7 **||** i **==** 8**)**

**{**

//do nothing

**}**

**else**

**{**

Point p **=** **new** Point**(**GRID\_SIZE **-** 1**,**i**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

// bottom wall

**for(**int i **=** 0**;**i **<** GRID\_SIZE**;**i**++)**

**{**

Point p **=** **new** Point**(**i**,**GRID\_SIZE **-**1**);**

walledCells**.**addElement**(**p**);**

**}**

//--------------------------------

//==================================

//Left Outer Grid

//==================================

//left top single

Point p1 **=** **new** Point**(**2**,**2**);**

walledCells**.**addElement**(**p1**);**

//left top square

**for(**int i **=** 1**;** i**<=**3**;** i**++)**

**{**

**for(**int j **=** 4**;**j **<=** 6**;**j**++)**

**{**

Point p **=** **new** Point**(**i**,**j**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

//left bottom square

**for(**int i **=** 1**;** i**<=**3**;** i**++)**

**{**

**for(**int j **=** 9**;**j **<=** 11**;**j**++)**

**{**

Point p **=** **new** Point**(**i**,**j**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

//left bottom single

Point p3 **=** **new** Point**(**2**,**13**);**

walledCells**.**addElement**(**p3**);**

//================================

//Left inner Grid

//================================

**for(**int i **=** 2**;**i **<=** 13**;**i**++)**

**{**

**if(**i **==** 7**||**i **==** 8**||**i **==** 12**)**

**{**

//do nothing

**}**

**else**

**{**

Point p **=** **new** Point**(**5**,**i**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

**for(**int i **=** 9**;** i**<=** 11**;**i**++)**

**{**

Point p **=** **new** Point**(**6**,**i**);**

walledCells**.**addElement**(**p**);**

**}**

**for(**int i **=** 2**;**i**<=** 6**;** i**++)**

**{**

**if(**i **==** 4**)**

**{**

//do nothing

**}**

**else**

**{**

Point p **=** **new** Point**(**7**,**i**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

//=====================================

// Right inner grid

//=====================================

**for(**int i **=** 8**;**i **<=** 10**;** i **=** i **+** 2**)**

**{**

**for(**int j **=** 2**;** j **<=** 13**;** j**++)**

**{**

**if(**j **==** 4 **||** j **==** 7 **||** j **==** 8 **||** **(**j **==** 10 **&&** i **==** 10**)||** **(**j **==** 12 **&&** i **==** 10**))**

**{**

//do nothing

**}**

**else**

**{**

Point p **=** **new** Point**(**i**,**j**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

**}**

//=====================================

//Right outer grid

//=====================================

//top right single

Point p4 **=** **new** Point**(**13**,**2**);**

walledCells**.**addElement**(**p4**);**

//top right square

**for(**int i **=** 12**;** i **<=** 14**;** i**++)**

**{**

**for(**int j **=** 4 **;** j **<=** 6**;** j**++)**

**{**

Point p **=** **new** Point**(**i**,**j**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

//Bottom Right Square

**for(**int i **=** 12**;** i **<=** 14**;** i**++)**

**{**

**for(**int j **=** 9 **;** j **<=** 11**;** j**++)**

**{**

Point p **=** **new** Point**(**i**,**j**);**

walledCells**.**addElement**(**p**);**

**}**

**}**

//Bottom right single

Point p5 **=** **new** Point**(**13**,**13**);**

walledCells**.**addElement**(**p5**);**

**}**

//========================================

//check if a point is one of the game walls

private boolean validPosition**(**Point a**)**

**{**

genWalls**();**

**if(**walledCells**.**contains**(**a**))**

**{**

**return** **false;**

**}**

**else** **return** **true;**

**}**

//========================================

//Generate a vector of free cells....i.e. cells avatar can move in

public void genFreeCells**()**

**{**

genWalls**();**

Point p**;**

//Backup walled cells

Vector**<**Point**>** backUp **=** **new** Vector**<**Point**>();**

Vector**<**Point**>** allCells **=** **new** Vector**<**Point**>();**

backUp **=** walledCells**;**

//make a vector with all grid point 16\*16

**for(**int i **=** 0**;**i **<** GRID\_SIZE**;**i**++)**

**{**

**for(**int j **=** 0**;**j **<** GRID\_SIZE**;**j**++)**

**{**

p **=** **new** Point**(**i**,**j**);**

allCells**.**addElement**(**p**);**

**}**

**}**

//remove all the elements in allCells that are in backUp

allCells**.**removeAll**(**backUp**);**

freeCells **=** allCells**;**

**}**

//----------------------------------

//Initialize the images

//---------------------------------

// sets up new buffer to hold images

private void initImages**()**

**{**

**if(**imagesDesired**)**

**{**

// buffer is cleared for each timestep, multiple images per timestep

buf **=** **new** ArrayList **<**BufferedImage**>** **();**

// block is only executed once (only read from disk once)

**if(**friezaStream **==** **null)**

**{**

**try**

**{** // use memory, not disk, for temporary images

ImageIO**.**setUseCache**(false);**

// read from disk

friezaStream **=** getClass**().**getResourceAsStream**(**IMG\_FRIEZA**);**

gokuStream **=** getClass**().**getResourceAsStream**(**IMG\_GOKU**);**

cloudStream **=** getClass**().**getResourceAsStream **(**IMG\_CLOUD**);**

bgStream **=** getClass**().**getResourceAsStream**(**IMG\_BG**);**

dragonballStream **=** getClass**().**getResourceAsStream**(**IMG\_DRAGONBALL**);**

deadStream **=** getClass**().**getResourceAsStream**(**IMG\_DEAD**);**

ginyuStream **=** getClass**().**getResourceAsStream**(**IMG\_GINYU**);**

superStream **=** getClass**().**getResourceAsStream**(**IMG\_SUPER**);**

lightStream **=** getClass**().**getResourceAsStream**(**IMG\_LIGHT**);**

cellStream **=** getClass**().**getResourceAsStream**(**IMG\_CELL**);**

friezaImg **=** javax**.**imageio**.**ImageIO**.**read**(**friezaStream**);**

gokuImg **=** javax**.**imageio**.**ImageIO**.**read**(**gokuStream**);**

cloudImg **=** javax**.**imageio**.**ImageIO**.**read**(**cloudStream**);**

bgImg **=** javax**.**imageio**.**ImageIO**.**read**(**bgStream**);**

dragonballImg **=** javax**.**imageio**.**ImageIO**.**read**(**dragonballStream**);**

deadImg **=** javax**.**imageio**.**ImageIO**.**read**(**deadStream**);**

ginyuImg **=** javax**.**imageio**.**ImageIO**.**read**(**ginyuStream**);**

superImg **=** javax**.**imageio**.**ImageIO**.**read**(**superStream**);**

lightImg **=** javax**.**imageio**.**ImageIO**.**read**(**lightStream**);**

cellImg **=** javax**.**imageio**.**ImageIO**.**read**(**cellStream**);**

// dimensions of jpg covering one square of the grid

imgwidth **=** friezaImg**.**getWidth**();**

imgheight **=** friezaImg**.**getHeight**();**

**}**

**catch(**IOException e**){}**

**}**

**}**

**}**

//-----------------------------------

//add images to the buffer

//------------------------------------

private void addImage**()**

**{**

//add walls to the wall vector

//genWalls();

**if** **(**imagesDesired**)**

**{**

BufferedImage img **=** **new** BufferedImage **((**imgwidth**\***GRID\_SIZE**),(**imgheight**\***GRID\_SIZE**),**BufferedImage**.**TYPE\_INT\_RGB**);**

// Draws background image and doesn't change it

img**.**createGraphics**().**drawImage**(**bgImg**,**0**,**0**,null);**

//=============================================

//Add the maze

//=============================================

**for(**int i **=** 0**;** i **<** GRID\_SIZE**;**i**++)**

**{**

**for(**int j **=** 0**;** j **<** GRID\_SIZE**;**j**++)**

**{**

Point p **=** **new** Point**(**i**,**j**);**

**if(**walledCells**.**contains**(**p**))**

**{**

img**.**createGraphics**().**drawImage **(** cloudImg**,** **(**imgwidth **\*** i**),** **(**imgheight **\*** j**),** **null** **);**

**}**

**}**

**}**

//if the dragonball wasn't collected

//then draw it on he map

**if(**dbCollected **==** 0**)**

**{**

img**.**createGraphics**().**drawImage **(** dragonballImg**,** **(**imgwidth **\*** dbPos**.**x**),** **(**imgheight **\*** dbPos**.**y**),** **null** **);**

**}**

//=============================================

**if** **(**friezaPos**.**equals**(**gokuPos**)** **||** ginyuPos**.**equals**(**gokuPos**)** **||** cellPos**.**equals**(**gokuPos**))**

**{**

//if Goku hasn't collected the db

//draw the dead img

**if(**dbCollected **==** 0**)**

**{**

img**.**createGraphics**().**drawImage **(** deadImg**,** **(**imgwidth **\*** gokuPos**.**x**),** **(**imgheight **\*** gokuPos**.**y**),** **null** **);**

**}**

//if goku is super saiyan

**if(**friezaPos**.**equals**(**gokuPos**)** **&&** friezaDied **==** **false** **&&** dbCollected **>** 0**)**

**{**

img**.**createGraphics**().**drawImage **(** superImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

**if(**cellDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** cellImg**,** **(**imgwidth **\*** cellPos**.**x**),(**imgheight **\*** cellPos**.**y**),** **null** **);**

**}**

**if(**ginyuDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** ginyuImg**,** **(**imgwidth **\*** ginyuPos**.**x**),(**imgheight **\*** ginyuPos**.**y**),** **null** **);** **}**

**if(**friezaDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** lightImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

friezaDied **=** **true;**

**}**

**}**

**else** **if(**ginyuPos**.**equals**(**gokuPos**)** **&&** ginyuDied **==** **false** **&&** dbCollected **>** 0**)**

**{**

img**.**createGraphics**().**drawImage **(** superImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

**if(**cellDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** cellImg**,** **(**imgwidth **\*** cellPos**.**x**),(**imgheight **\*** cellPos**.**y**),** **null** **);**

**}**

**if(**friezaDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** friezaImg**,** **(**imgwidth **\*** friezaPos**.**x**),** **(**imgheight **\*** friezaPos**.**y**),** **null** **);**

**}**

**if(**ginyuDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** lightImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

ginyuDied **=** **true;**

**}**

**}**

**else** **if(**cellPos**.**equals**(**gokuPos**)** **&&** cellDied **==** **false** **&&** dbCollected **>** 0**)**

**{**

img**.**createGraphics**().**drawImage **(** superImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

**if(**ginyuDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** ginyuImg**,** **(**imgwidth **\*** ginyuPos**.**x**),(**imgheight **\*** ginyuPos**.**y**),** **null** **);**

**}**

**if(**friezaDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** friezaImg**,** **(**imgwidth **\*** friezaPos**.**x**),** **(**imgheight **\*** friezaPos**.**y**),** **null** **);**

**}**

**if(**cellDied **==** **false)**

**{**

img**.**createGraphics**().**drawImage **(** lightImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

cellDied **=** **true;**

**}**

**}**

**}**

**else**

**{**

**if(**dbCollected **==** 0**)**

**{**

img**.**createGraphics**().**drawImage **(** gokuImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

**}**

**else**

**{**

img**.**createGraphics**().**drawImage **(** superImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

**}**

// only draw ginyu and frieza if they exist

**if(**friezaHealth **==** 1**)**

**{**

img**.**createGraphics**().**drawImage **(** friezaImg**,** **(**imgwidth **\*** friezaPos**.**x**),** **(**imgheight **\*** friezaPos**.**y**),** **null** **);**

**}**

**if(**ginyuHealth **==** 1**)**

**{**

img**.**createGraphics**().**drawImage **(** ginyuImg**,** **(**imgwidth **\*** ginyuPos**.**x**),(**imgheight **\*** ginyuPos**.**y**),** **null** **);**

**}**

**if(**cellHealth **==** 1**)**

**{**

img**.**createGraphics**().**drawImage **(** cellImg**,** **(**imgwidth **\*** cellPos**.**x**),(**imgheight **\*** cellPos**.**y**),** **null** **);**

**}**

**}**

//if the dB is collected

//check who we need to draw and where

**if(**dbCollected **==** 1**)**

**{**

**if(**friezaHealth **==** 1 **&&** **!**friezaPos**.**equals**(**gokuPos**))**

**{**

img**.**createGraphics**().**drawImage **(** friezaImg**,** **(**imgwidth **\*** friezaPos**.**x**),** **(**imgheight **\*** friezaPos**.**y**),** **null** **);**

**}**

**if(**ginyuHealth **==** 1 **&&** **!**cellPos**.**equals**(**gokuPos**))**

**{**

img**.**createGraphics**().**drawImage **(** ginyuImg**,** **(**imgwidth **\*** ginyuPos**.**x**),(**imgheight **\*** ginyuPos**.**y**),** **null** **);**

**}**

**if(**cellHealth **==** 1 **&&** **!**cellPos**.**equals**(**gokuPos**))**

**{**

img**.**createGraphics**().**drawImage **(** cellImg**,** **(**imgwidth **\*** cellPos**.**x**),(**imgheight **\*** cellPos**.**y**),** **null** **);**

**}**

**if(**gokuHealth **>** 0**)**

**{**

img**.**createGraphics**().**drawImage **(** superImg**,** **(**imgwidth **\*** gokuPos**.**x**),(**imgheight **\*** gokuPos**.**y**),** **null** **);**

**}**

**}**

buf**.**add**(**img**);**

**}**

**}**

//====== World must respond to these methods: ==========================================================

// newrun(), endrun()

// getstate(), takeaction()

// getscore()

//======================================================================================================

//--------------------------------

//Initializes a new run of the world

//--------------------------------

public void newrun**()** **throws** RunError

**{**

//Create Points to store position of cops + robbers

friezaPos **=** **new** Point**();**

ginyuPos **=** **new** Point**();**

gokuPos **=** **new** Point**();**

cellPos **=** **new** Point**();**

//Place the db in top right corner

dbPos **=** randomPosition**();**

//Reset all values

timestep **=** 0**;**

numCaught **=** 0**;**

//a count of the dragonballs collected

dbCount **=** 0**;**

//give goku 3 lives

gokuHealth **=** 3**;**

ginyuHealth **=** 1**;**

friezaHealth **=** 1**;**

cellHealth **=** 1**;**

//use initializer

initPos**();**

genWalls**();**

//Headers for score fields

scoreCols **=** **new** LinkedList**<**String**>();**

scoreCols**.**add**(**"numCaught"**);**

scoreCols**.**add**(**"gokuHealth"**);**

scoreCols**.**add**(**"DragonBall Count"**);**

scoreCols**.**add**(**"NumBadGuys Killed"**);**

**}**

public void endrun**()** **throws** RunError

**{**

**}**

//====== Definition of state: ===========================================================================

// Constructs a string to describe the curent world state

//======================================================================================================

public State getstate**()** **throws** RunError

**{**

String x **=** String**.**format **(**"%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d"**,** friezaPos**.**x**,**friezaPos**.**y**,**gokuPos**.**x**,**gokuPos**.**y**,**dbPos**.**x**,**dbPos**.**y**,**

ginyuPos**.**x**,**ginyuPos**.**y**,**dbCollected**,**friezaHealth**,**ginyuHealth**,**cellPos**.**x**,**cellPos**.**y**,**cellHealth**);**

**return** **new** State **(**x**);**

**}**

//=========================================================================================================

//Mind.takeaction() constructs a string to describe the action.

//=========================================================================================================

public State takeaction **(**Action action**)** **throws** RunError

**{**

// If run with images off, imagesDesired = false and this does nothing.

initImages**();**

**if(**run1 **==** **true)**

**{**

addImage**();**

run1 **=** **false;**

**}**

//parse the action

String s **=** action**.**toString**();**

// parsed into a[0], a[1], ...

String**[]** a **=** s**.**split**(**","**);**

// ignore any other fields

int i **=** Integer**.**parseInt**(**a**[**0**]);**

int j **=** Integer**.**parseInt**(**a**[**1**]);**

int k **=** Integer**.**parseInt**(**a**[**2**]);**

int m **=** Integer**.**parseInt**(**a**[**3**]);**

// take the action

**if(**friezaHealth **>** 0**)**

**{**

move**(**friezaPos**,**i**);**

**}**

**if(**ginyuHealth **>** 0**)**

**{**

move**(**ginyuPos**,**k**);**

**}**

**if(**cellHealth **>** 0**)**

**{**

move**(**cellPos**,**m**);**

**}**

//intermidiate image

addImage**();**

//if goku loses all health end the game

//or if the baddies were all killed

**if(**gokuHealth **<=** 0 **||** numBadGuys **<=** 0**)**

**{**

timestep **=** 100**;**

**}**

**if(**friezaPos**.**equals**(**gokuPos**)** **||** ginyuPos**.**equals**(**gokuPos**)** **||** cellPos**.**equals**(**gokuPos**))**

**{**

**if((**dbCollected **==** 0 **&&** friezaPos**.**equals**(**gokuPos**)** **&&** friezaHealth **>** 0**)**

**||** **(**dbCollected **==** 0 **&&** ginyuPos**.**equals**(**gokuPos**)** **&&** ginyuHealth **>** 0**)**

**||** **(**dbCollected **==** 0 **&&** cellPos**.**equals**(**gokuPos**)** **&&** cellHealth **>** 0**))**

**{**

numCaught**++;**

gokuHealth **=** gokuHealth **-** 1**;**

initPos**();**

**}**

//if we have all the db's goku can kill enemies

**else**

**{**

**if(**friezaPos**.**equals**(**gokuPos**)** **&&** friezaHealth **>** 0**)**

**{**

friezaHealth **=** friezaHealth **-** 1**;**

numBadGuys**--;**

//if frieza dies move him to an invalid location

addImage**();**

friezaPos**.**move**(**0**,**0**);**

**}**

**if(**ginyuPos**.**equals**(**gokuPos**)** **&&** ginyuHealth **>** 0**)**

**{**

ginyuHealth **=** ginyuHealth **-** 1**;**

numBadGuys**--;**

addImage**();**

ginyuPos**.**move**(**0**,**0**);**

**}**

**if(**cellPos**.**equals**(**gokuPos**)** **&&** cellHealth **>** 0**)**

**{**

cellHealth**--;**

numBadGuys**--;**

addImage**();**

cellPos**.**move**(**0**,**0**);**

**}**

**}**

**}**

// move the robber

**else**

**{**

move**(**gokuPos**,**j**);**

**if(**gokuPos**.**equals**(**dbPos**))**

**{**

//GAIN 1 life

gokuHealth **=** gokuHealth **+** 1**;**

//reset the dragonball

dbPos **=** randomPosition**();**

dbCount **=** dbCount **+**1**;**

//if we colleced enough db's become super saiyan

//set to one for use on w2mind

**if(**dbCount **==** 1**)**

**{**

dbCollected **=** 1**;**

**}**

**}**

addImage**();**

**if** **(**friezaPos**.**equals**(**gokuPos**)** **||** ginyuPos**.**equals**(**gokuPos**)** **||** cellPos**.**equals**(**gokuPos**))**

**{**

**if((**dbCollected **==** 0 **&&** friezaPos**.**equals**(**gokuPos**)** **&&** friezaHealth **>** 0**)**

**||(**dbCollected **==** 0 **&&** ginyuPos**.**equals**(**gokuPos**)** **&&** ginyuHealth **>** 0**)**

**||(**dbCollected **==** 0 **&&** cellPos**.**equals**(**gokuPos**)** **&&** cellHealth **>** 0**))**

**{**

numCaught**++;**

// caught due to our action, not robber's action

gokuHealth **=** gokuHealth **-** 1**;**

addImage**();**

initPos**();**

**}**

//if we have all the db's goku can kill enemies

**else**

**{**

**if(**friezaPos**.**equals**(**gokuPos**)** **&&** friezaHealth **>** 0**)**

**{**

friezaHealth **=** friezaHealth **-** 1**;**

numBadGuys**--;**

addImage**();**

friezaPos**.**move**(**0**,**0**);**

**}**

**if(**ginyuPos**.**equals**(**gokuPos**)** **&&** ginyuHealth **>** 0**)**

**{**

ginyuHealth **=** ginyuHealth **-** 1**;**

numBadGuys**--;**

addImage**();**

ginyuPos**.**move**(**0**,**0**);**

**}**

**if(**cellPos**.**equals**(**gokuPos**)** **&&** cellHealth **>** 0**)**

**{**

cellHealth**--;**

numBadGuys**--;**

addImage**();**

cellPos**.**move**(**0**,**0**);**

**}**

**}**

**}**

**}**

timestep**++;**

//there will be no loop around if run is finished so print image

**if(**runFinished**())**

addImage**();**

**return** getstate**();**

**}**

//==========================================================================================================================

// get game score

//==========================================================================================================================

public Score getscore**()** **throws** RunError

**{**

int numBadGuysKilled **=** 3 **-** numBadGuys**;**

String s **=** String**.**format **(**"%d,%d,%d,%d"**,** numCaught**,** gokuHealth**,** dbCount**,**numBadGuysKilled**);**

// Setting finished = true will end the run.

// N.B. This is the only way the World has to tell the underlying w2m system to stop the run.

boolean finished **=** **(** timestep **>=** MAX\_STEPS **);**

List **<**Comparable**>** values **=** **new** LinkedList **<**Comparable**>** **();**

values**.**add**(**numCaught**);**

values**.**add**(**gokuHealth**);**

values**.**add**(**dbCount**);**

values**.**add**(**numBadGuysKilled**);**

**return** **new** Score**(**s**,**finished**,**scoreCols**,**values**);**

**}**

//return the images of the world

public ArrayList**<**BufferedImage**>** getimage**()** **throws** RunError

**{**

**return** buf**;**

**}**

**}**