**Detailed Documentation**

**Class: Game**

public class Game

**Members**

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| List<GameObject> | go | List for keeping track of moving GameObjects |
| List<OffGridGO> | nongo | List for OffGridGOs, since they are not stored in the Grid |
| Map<Item, Integer> | inventory | The player's inventory, modelled as a map of items to count of how many |
| GameObject | player | Reference to the player's GameObject |
| GameGrid | gg | Object that models the game world. It contains an actual 'grid' of GameObjects |
| HitboxGrid | hbg | Object that keeps within it currently loaded Hitboxes |
| MainFrame | gameWindow | The window in which the game is displayed |
| DrawingPanel | gameScreen | Panel that sits within MainFrame, which everything is drawn to (this is how Swing works) |
| PauseMenu | menu | Implementation of a MenuSystem packaged in a single class to act as the game's Pause menu. |
| Keys | ctrl | Class that gets and stores information about player input |
| static TextboxWriter | txt | System for writing messages to the screen. Static so it can be accessed with reference to an object |

**Methods**

**Constructors**

**Game()**

Initialises members, and the game as a whole. Loads level names from Levels directory so they can be used later, load the first level to the Grid. The MainFrame is setup so it can display the game, and so user input is directed to appropriate controlling classes. It is also used to hard-code loading objects into the game, to get around the limitations of the Loading system.

**static void Main(String[] args)**

Entry point to the program. Creates a Game object that represents (the current instance of) the game.  
This method holds the game loop; The game is updated and is re-drawn to screen. In order to keep the frame-rate consistent, The time the last frame took is used to find out how long to sleep the thread so that the target time is reached for the target framerate. The frame-rate display is also updated.

**void update()**

This method handles frame updates for the entire game.  
It also handles what to do based on the state of the Controller;

* Switch between fullscreen and windowed
* What to do if paused (nothing)
* Advance text if a dialogue is open, or close it if the text is finished
* Check if movement or attacking is allowed, then process it if so.

Attacking;

* Get the attacking input
* Generate appropriate Hitboxes or OffGridGOs
* Add the generated objects to the respective list
* Update controller's cooldown

The update methods on each type of Object being kept track of.  
(Dynamic) GameObject:

* If the the GO is within 20 grid spaces of the player then update it
* Use GameGrid's move method to move GOs into the correct place
* If the GO is an enemy, handle its attacks
* If the GO is 'dead' then remove it from the list and the grid.

OffGridGOs:

* Handle collisions with grid objects, for example if it is an item, it can be picked up by the player
* Remove if not 'alive'

Hitboxes:

* Update hitboxes based on player position
* For spreading hitboxes, find them in hitbox list
* get the hitboxes they have spawned and add them to the list

**static <T> void addItem(Map<T, Integer> map, T item)**

Utility method used for adding items to inventory, but genericised.

| **Argument** | **Description** |
| --- | --- |
| map | the map to add to |
| item | the item (of any type) to add to the map |

Add item to map, increment the number or set it to 0 if new item.

**static <T> void removeItems(Map<T, Integer> map, T item, int number)**

Utility method used for removing items from inventory, but genericised.

| **Argument** | **Description** |
| --- | --- |
| map | the map to add to |
| item | the item (of any type) to add to the map |
| number | the number to be added |

If the item exists, decrement its count. If the number goes below 1 then remove it from the map.

# Class: MainFrame

class MainFrame  
extends JFrame

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| boolean | fullscreen | in fullscreen mode or not? |

## Methods

### Constructors

#### MainFrame()

Set default parameters (size, close operation etc). Set fullscreen if fullscreen is enabled.

# Class: DrawingPanel

class DrawingPanel extends JPanel

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| Game | game | reference to the game's Game object |
| double | fps | the framerate that the game is currently running at |

## Methods

### Constructors

#### DrawingPanel(Game g)

| **Argument** | **Description** |
| --- | --- |
| g | reference to the game's Game object. Should be passed from the game object itself |

Initialise Game parameter.

### void paintComponent(Graphics g)

Overridden from JPanel

Draw the game scene to the screen. The scene is a 13x13 grid. The order of events described below is equivalent to the order in which things are rendered.

| **Argument** | **Description** |
| --- | --- |
| g | graphics object which the drawing is applied to. Should be from a JFrame |

1. Initialisation  
   Use the screen's width and height to determine the scale at which to draw the image. Find the position of the top-left of the game view such that it is centred in the window, and translate g to this position. Store this as the default position.
2. Screen Scroll  
   Compute the position the game view should be in so that the player is in the middle of the grid. Unless the player is at the edge of the screen, then keep the edge of the game grid aligned with the edge of the view.
3. Rendering the game Iterate through every grid square that is on-screen. First draw a 'background' tile so that tiles that have transparency have a background. Then draw the current GO.  
   Drawing non-grid objects: Draw each OffGridGO that is on-screen, making sure it is in its exact position and rotation.  
   Hitboxes are also drawn if they have their own sprite (this is handled by the Hitbox system).
4. UI
   * If enabled, every hitbox is visible with a red square, overlaid if it already has a sprite.
   * If enabled, grid squares are drawn
   * Draw the player health bar
   * If debug enabled:
     + draw FPS display
     + draw player coordinates
     + draw screen coordinates
     + draw the Input and Action buffers (see Keys for detailed explanation of buffers)
     + draw the direction the player is facing (as an x and y direction)
   * If message dialogue is open
     + Draw a box on screen for the dialogue
     + Draw the appropriate text on the box
   * If the menu is open
     + Draw a box for the menu background
     + Draw the menu

**Class: Keys**

public Class Keys  
extends KeyAdapter

**Members**

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| boolean | pause | is the game paused |
| boolean | text\_pause | is the game paused because of a dialogue |
| boolean | close\_text | signal to close text dialogue |
| boolean | switchFS | signal to switch between fullscreen and windowed |
| boolean | grid | is the grid displayed |
| boolean | debug | is debug mode on |
| boolean | interact | signal that player is attempting to interact |
| int | vert | vertical movement - positive for up, negative for down |
| int | hor | horizontal movement - positive for right, negative for left |
| int | xdir | facing direction in x-axis - positive for right, negative for left |
| int | ydir | facing direction in y-axis - positive for up, negative for down |
| long | time\_of\_last\_movement | the time (in nanoseconds) that the player last moved |
| long | time\_of\_last\_action | the time (in nanoseconds) that the player last did an action |
| long | action\_lag | the duration during which the player cannot act |
| long[] | lag\_table | the time until the player can act again for each action |
| int | input\_buffer | store the key code of movement input if the player is already moving |
| int | action\_buffer | store the key code of action input if the player is already acting |
| int | attack | the id of the attack |

**Methods**

**Constructors**

**Keys()**

Initialise members to default values. For times this is the current time.

**void keyPressed(KeyEvent e)**

Handle KeyEvents - when the user presses a key.

| **Argument** | **Description** |
| --- | --- |
| e | Information about keyboard input |

For actions, set the buffer to the action corresponding to the key. This is so that the action is executed when the key is released.  
For movement, if there is currently no movement input, then the corresponding direction can be set. Otherwise, set the movement buffer to the new key.  
For other keys, the corresponding flag can be set.

**void keyReleased(KeyEvent e)**

Handle KeyEvents - when the user releases a key.

| **Argument** | **Description** |
| --- | --- |
| e | Information about keyboard input |

If the key released is the one in the buffer, then clear the buffer.  
For movement, if the key released is the one corresponding the current movement, then revert to the default value. Then check the buffer input.  
For attacks, the attack id is set, buffer is cleared and the lag is set. For interaction command, if no dialogue is open, set flag to true, otherwise it is false. If a dialogue is open, 'close\_text' is set to true, which will either advance the text, or close the dialogue depending on the situation.

**void checkBuffer()**

Used to carry out buffer input when the current action's key is released.

If the buffer isn't empty and the buffer's input corresponds to movement, then set that movement.

# Class: Controller

public class Controller

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| static GameObject | playerref | Reference to the current game's player |
| GameObject | me | reference to the GameObject that the Controller is controlling |
| int | xmove | the next move in x-direction |
| int | ymove | the next move in y-direction |
| int | xdir | facing direction on x-axis |
| int | ydir | facing direction on y-axis |
| int | move\_lag | time to wait between moves (frames) |
| int | lag\_timer | count how long since last move (frames) |

## Methods

### Constructors

#### Controller()

Initialise members to default value (0, apart from move\_lag, which by default is 30)

### void setGameObjectRef(GameObject gameObject)

Set the reference of the GameObject to be controlled by the controller.

| **Argument** | **Description** |
| --- | --- |
| gameObject | The GameObject to be controlled by the controller |

### void update()

This class is not intended to be used, but extended. As such, this update method does nothing.

# Class: VBAIController

class VBAIController extends Controller

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| Random | random | For generating random numbers - to pick actions |
| boolean | moving | Am I already in motion |

## Methods

### Constructors

#### VBAIController()

Call super-constructor. Pick random x direction, then if x direction was 0, pick a random y direction (that is not 0).

### void update()

Overridden from Controller.  
Update the state of the Controller.

Randomly decide to switch between moving or not.  
Randomly pick a direction to face.  
If in moving state and enough time has elapsed then move in the facing direction (by 1)

# Class: BasicEnemyController

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| boolean | enemy\_spotted | Whether the player is visible or not |
| Hitbox | myHB | Store a hitbox so it can be accessed from Game |

## Methods

### Constructors

#### BasicEnemyController(GameObject gameObject)

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| GameObject | me | reference to the GameObject that the Controller is controlling |

Call default super-constructor. Initialise GameObject reference. Set move\_lag to 60 frames.

### void update()

Overridden from Controller.  
Update the state of the Controller.

Compute if the player is 'visible' by finding if the distance to the player is small enough.  
If enough time has elapsed since last action:  
Try to set the next move to move towards the player. If the player is adjacent then don't move.  
In that case, attempt to attack the player.  
If the player is not visible call the super-class's method - i.e. move randomly.

# Class: GameGrid

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| GameObject[][] | grid | the grid that holds GameObjects (2D array). Index represents position. |
| List | list | list of (dynamic, static objects not required) GameObjects in the grid. Used for quicker access. |
| int | width | the width of the grid |
| int | height | the height of the grid |

## Methods

### Constructors

#### GameGrid(int width, int height)

| **Argument** | **Description** |
| --- | --- |
| width | the width of the grid |
| height | the height of the grid |

Save the width and height of the grid and create a new grid with these dimensions.

### boolean move(GameObject go, int hor\_move, int vert\_move)

Move an object in the grid by the required amount. Returns true if move is completed, false if it cannot be.

| **Argument** | **Description** |
| --- | --- |
| go | the GameObject to move |
| hor\_move | the amount to move horizontally |
| vert\_move | the amount to move vertically |

If the move would put the object outside of the grid or the new position is the same as the old one, then stop.  
If the new position is empty or the GameObject collison returns true, then update the grid and the object.

# Class: GameObject

public class GameObject

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| int | x | x position in grid |
| int | y | y position in grid. note: top is 0, so +ve y is down |

## Methods

### Constructors

#### GameObject()

Initialise x and y to default value (0)

#### GameObject(int x, int y)

| **Argument** | **Description** |
| --- | --- |
| x | x position in grid |
| y | y position in grid |

Initialise x and y to given values.

### void move(int h, int v)

| **Argument** | **Description** |
| --- | --- |
| h | horizontal move |
| v | vertical move, where +ve is up, -ve is down |

### void paint(Graphics g)

Default paint method if extensions don't have their own.  
Draws a blue square in that fills the grid space.

### void hit(Hitbox hb)

What to do when hit by a hitbox.

| **Argument** | **Description** |
| --- | --- |
| hb | The hithbox that this GameObject has been hit by. |

Default GameObjects shouldn't react to being hit.

### boolean collide(GOT other)

Define how to collide with other GameObjects

| **Argument** | **Description** |
| --- | --- |
| other | The GameObject this object is colliding with |

Never allow this object to tbe passed over.

### void update()

Update this GameObject by a frame.

By default, do nothing.

### static double dist(GameObject a, GameObject b)

Calculate distance between two GameObjects

| **Argument** | **Description** |
| --- | --- |
| a | First GameObject |
| b | Second GameObject |

square root of [(x2 - x1)^2 + (y2 - y1)^2]

# Class: Wall

class Wall  
extends GameObject

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| boolean | solid | Can hitboxes pass through it or not |

## Methods

### void paint(Graphics g)

Render a wall with the wall sprite.

### void hit(Hitbox hb)

Hitbox is used if the Wall is 'solid'

# Class: Chara

class Chara  
extends GameObject

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| enum | char\_class | which classes are there: (none, player or enemy) |
| char\_class | cclass | Which class does this belong to. |
| int | max\_hp | the maximum hp the Chara can have |
| int | atk | the attack stat, used to calculate damage dealt |
| int | def | the defense stat, used to calculate damage taken |
| double | hp | the amount of health currently |
| boolean | dead | is the Chara dead? |
| Image | sprite | the sprite to render |
| String | prefab | the name of the prefab used to make this object |
| Map<Item, Integer> | inventory | Inventory of items, and number of them. |
| static Map<String, Chara> | prefabs | Map of Name->Prefab |

## Methods

### Constructors

#### Chara(int x, int y, char\_class cclass, int max\_hp, int atk, int def, Image sprite)

| **Argument** | **Description** |
| --- | --- |
| x | x position in grid |
| y | y position in grid |
| cclass | Which class does this belong to. |
| max\_hp | the maximum hp the Chara can have |
| atk | the attack stat, used to calculate damage dealt |
| def | the defense stat, used to calculate damage taken |
| sprite | the sprite to render |

Initialise members to given values.

### void setInventory(Map<Item, Integer> inventory)

| **Argument** | **Description** |
| --- | --- |
| inventory | Inventory of items, and number of them. |

Set inventory member to be the given Map.

### static Chara newChara(Chara prefab, int x, int y)

Make a new object from a prefab Return value: a new Chara object as defined by the given Chara

| **Argument** | **Description** |
| --- | --- |
| prefab | Object to copy values from |
| x | x position |
| y | y position |

Create a new Chara with the parameters of the given one. Also intialise the Controller.

### void update()

Overridden from GameObject

If HP is 0 (or less) then the Chara is dead.  
Update the controller.

### void hit(Hitbox hb)

If the char\_class is the same as the hitbox's owner then don't hit.  
Calculate damage and take damage from health.  
Then the hitbox is 'used'

# Class: Sign

class Sign  
extends Wall

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| String | text | The text 'written on' the sign |

## Methods

### Constructors

#### Sign(int x, int y, String text)

| **Argument** | **Description** |
| --- | --- |
| x | x position in grid |
| y | y position in grid |
| text | The text 'written on' the sign |

# Class: Door

## Methods

### Constructors

#### Door(int x, int y)

Pass to GameObject constructor.

### boolean collide(GOT other)

Overridden from super class.

Only let another object through if it holds a key that can open this Door.

# Other Classes

Water, WindowWall extends Wall - not solid

All extensions are rendered with their own sprite.

**Class: OffGridGO**

public class OffGridGO  
extends GameObject

**Members**

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| double | offX | offset from middle of grid position, as fraction of a grid square length. From -0.5 to 0.5 - in x direction |
| double | offY | same as above for y direction |
| double | velx | velocity in x direction |
| double | vely | velocity in y direction |
| Image | sprite | the sprite rendered to represent this object |
| boolean | airborne | is the object airborne (can go over things) |
| boolean | alive | is it alive |

**Methods**

**Constructors**

**OffGridGO(int x, int y, double xoffset, double yoffset)**

| **Argument** | **Description** |
| --- | --- |
| x | x position |
| y | y position |
| xoffset | offset from middle of grid position on x axis |
| yoffset | offset from middle of grid position on y axis |

**void fixGridPos()**

Fix position variables so that offsets don't go out of range.

For x and y offsets, if they are out of range, truncate them and change the grid position, then remove the integer part from the offset.

**void offGridMove(double xmove, double ymove)**

Move the object by changing the off-grid position

| **Argument** | **Description** |
| --- | --- |
| xmove | how much to change off-grid x position |
| ymove | how much to change off-grid y position |

Add given values to offsets, then fix the position back to grid.

**OffGridGO setVelocity(double vx, double vy)**

Returns: This object with the velocity set.

| **Argument** | **Description** |
| --- | --- |
| velx | x velocity |
| vely | y velocity |

Set the velocities to the given values.

**void update()**

Overridden from GameObject

Move the object by the amount given by velocities.

**void paint(Graphics g)**

Overridden from GameObject

If sprite is set draw the sprite. Otherwise the default for OffGridGOs is a cyan rectangle smaller than grid square.

**Class: Item**

class Item  
extends OffGridGO

**Members**

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| String | name | The name of this item |

**Methods**

**Constructors**

**See super-class constructors**

**Item getThisWithName(String name)**

Returns: This object with name set.

| **Argument** | **Description** |
| --- | --- |
| name | The name to set |

Set the name as given value. Attempt to set sprite to one with same name.

**boolean equals(Object obj)**

Overridden from super-class.  
Returns: If this object is equal to another.

| **Argument** | **Description** |
| --- | --- |
| obj | The object to compare to. |

Two items are equal if they have the same name.

**static Image grabImage(Item item)**

Attempt to get sprite with same name as an item.

| **Argument** | **Description** |
| --- | --- |
| item | The item with name to use |

If the item has no name, return no Image.  
If there is not corresponding sprite, return the 'null item' sprite.  
Otherwise, can return the corresponding sprite.

**Class: Key**

class Key  
extends Item

**Members**

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| Door | door | The door that this key opens |

**Methods**

**Constructors**

**See super-class constructors**

**Key setDoor(Door door)**

Returns: This object.

| **Argument** | **Description** |
| --- | --- |
| door | The door that this key opens |

Set the door that this key opens.

**boolean equals(Object obj)**

Overridden from super-class.  
Returns: If this object is equal to another.

| **Argument** | **Description** |
| --- | --- |
| obj | The object to compare to. |

Two items are equal if they open the same Door.

**Class: Projectile**

class Projectile  
extends OffGridGO

**Members**

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| Hitbox | hitbox | A hitbox that is associated with this projectile |

**Methods**

**Constructors**

**See super-class constructors**

**Projectile setHitbox(Hitbox hitbox)**

Returns: This object.

| **Argument** | **Description** |
| --- | --- |
| hitbox | The hitbox to set. |

Sets the hitbox member to given object.

**void update()**

Overridden from super-class.

If the hitbox is no longer alive, neither should this object.  
Set the hitbox position to be the (integer) position of this object.

**Class: Grenade**

class Grenade  
extends Projectile

**Members**

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| int | hangtime | how long it stays 'airborne' - in frames |
| int | detonator | how long until it 'explodes' - in frames |
| int | lifetime | how long it has been alive for - in frames |

**Methods**

**Constructors**

**Grenade(int x, int y, double xoffset, double yoffset, double velx, double vely, int detonator\_length)**

| **Argument** | **Description** |
| --- | --- |

Initialise members to the given values. Airborne by default starts as true. Hangtime is set to be the magnitude of the resultant vector of the velocities, and scaled up.

**void update()**

Overridden from super-class.

Calls super-class update.  
Increment lifetime counter. If its life has exceeded hangtime that it is set to no airborne.  
If it is not airborne, it is stopped from moving.  
If its life has exceeded the detonator time, then it is set to not be alive.

# Class: HitboxGrid

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| int | width | width of the GameGrid this is associated with |
| int | height | height of the GameGrid |
| List | hb | list of hitboxes to store in |

## Methods

### Constructors

#### HitboxGrid(int width, int height)

| **Argument** | **Description** |
| --- | --- |
| width | width of the GameGrid this is associated with |
| height | height of the GameGrid |

Initialise members to given value, empty list.

### boolean add(Hitbox newh)

Add a hitbox to the list. Returns: If hitbox was successfully added.

| **Argument** | **Description** |
| --- | --- |
| newh | A hit box to add to the list |

If the position is outside of the grid then don't add.  
Check if the position already contains a hitbox. If no, add, if yes only add if the power is greater.

### boolean add(int x, int y, int power, int lifetime, int xspeed, int yspeed)

Add a hitbox to the list. Returns: If hitbox was successfully added.

| **Argument** | **Description** |
| --- | --- |
| x | x position |
| y | y position |
| power | the stat used to determine damage output |
| lifetime | how long the hitbox is active for |
| xspeed | how fast the hitbox moves in x direction |
| yspeed | how fast the hitbox moves in y direction |

Create a new hitbox with the given parameters. If the position is outside of the grid then don't add.  
Check if the position already contains a hitbox. If no, add, if yes only add if the power is greater.

### void update(int playerx, int playery)

Update hitboxes relative to player position.

| **Argument** | **Description** |
| --- | --- |
| playerx | The player's x position |
| playery | The player's y position |

Update every hitbox in the list, passing player x and y position.  
After updating, remove it if it has been set as inactive.

### void paint(Graphics g, int sx, int sy, boolean debug)

Draw hitboxes to screen.

| **Argument** | **Description** |
| --- | --- |
| g | Graphics object from Frame |
| sx | screen x - the x position of the game view |
| sy | screen y - the x position of the game view |
| debug | whether debug mode is enabled |

If it has a sprite set, draw each hitbox at the right place on screen.  
If debug mode is enabled, draw a translucent square to represent the hitbox.

# Class: Hitbox

class Hitbox  
implements Comparable<Hitbox>

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| int | x | x position |
| int | y | y position |
| int | age | time since creation in frames |
| int | lifetime | how long to live in frames |
| double | power | the stat used to determine damage output |
| double | xspeed | how many frames to update position in x direction |
| double | yspeed | how many frames to update position in y direction |
| boolean | active | is this hitbox active |
| boolean | multihit | does this hitbox stay active if it hits something |
| Chara.char\_class | owner\_class | which class does the creator of this hitbox belong to |
| Image | sprite | sprite to draw |
| double | sprite\_rotation | rotate sprite by this angle (in degrees) |

## Methods

### Constructors

#### Hitbox(int x, int y, double power, int lifetime, double xspeed, double yspeed, Chara.char\_class owner)

| **Argument** | **Description** |
| --- | --- |
| x | x position |
| y | y position |
| age | time since creation in frames |
| lifetime | how long to live in frames |
| power | the stat used to determine damage output |
| xspeed | how many frames to update position in x direction |
| yspeed | how many frames to update position in y direction |
| active | is this hitbox active |
| multihit | does this hitbox stay active if it hits something |
| owner\_class | which class does the creator of this hitbox belong to |

Initialise members to given values. Age is set to 0.

### int compareTo(Hitbox other)

Defines such that Hitboxes are sorted by their position, by x first. Returns positive if larger, negative if smaller, 0 if equal.

| **Argument** | **Description** |
| --- | --- |
| other | The hitbox to compare this one to |

Compare x, y, age in that order. If different return this - other.  
If 'this' should come first it is 'less' - further top left or younger.

### void update(int playerx, int playery)

Update relative to player position.

| **Argument** | **Description** |
| --- | --- |
| playerx | The player's x position |
| playery | The player's y position |

Increment age. Move by one in appropriate direction(s) if it is the right time.

### void used

If this is not a multihit hitbox then set as inactive.

# Class: SpreadHitbox

class SpreadHitbox extends Hitbox

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| int | spread | the radius of the spread - as number of squares from centre |
| int | dimFac | percentage of power kept when spreading |
| int | iteration | time since creation in frames |
| boolean | square | is the spread in square pattern |
| boolean | dim | should power be diminished when spreading |

## Methods

### Constructors

#### SpreadHitbox(int x, int y, double power, int lifetime, Chara.char\_class owner, int spread\_radius, boolean square\_spread, boolean diminish\_power, int diminish\_factor)

| **Argument** | **Description** |
| --- | --- |
| x | x position |
| y | y position |
| power | the stat used to determine damage output |
| lifetime | how long to live in frames |
| owner | which class does the creator of this hitbox belong to |
| spread\_radius | the radius of the spread - as number of squares from centre |
| square\_spread | is the spread in square pattern |
| diminish\_power | does this hitbox stay active if it hits something |
| diminsh\_factor | should power be diminished when spreading |

Initialise members with given values.  
If radius given is less than 1, set to 1.  
If told not to diminish power, or the given factor is outside range 0-100 the set it to 100 (no diminishing)

### List<Hitbox> spread()

Process which squares to spread to.  
Returns: List of new hitboxes - the result of the spread.

Only spread if the desired radius has not yet been reached.  
Calculate diminished power - take percentage of base power.  
Trace square around centre, creating a new hitbox at each place.  
Increment iteration when completed.

# Class: LevelLoader

public class LevelLoader

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| static Map<String, String> | levelNames | maps Name to file name |
| static final String | EXTENSION | file extension for level files |
| static String | directory | the directory to look in for levels |

## Methods

### Constructors

#### LevelLoader has no constructors

### loadLevelNames(String[] names, String[] filenames)

| **Argument** | **Description** |
| --- | --- |
| names | names of levels |
| filenames | the file names of the level files |

Initialise the levelNames map. For each pair of given names and filenames, add them to the map.

### static void loadLevelNames()

Initialise empty levelNames map.

### static GameGrid loadLevel(String levelName)

Returns: A GameGrid that contains the loaded level.

| **Argument** | **Description** |
| --- | --- |
| levelName | the name of the level to load |

Read the length and width from the file and make a new grid with these dimensions.  
For every character in the file, attempt to find the corresponding GameObject using *getGameObject* and place it in the grid.  
After the grid is specification of other objects. Parse these lines and produce a GameObject with the specified parameters. Add them to the GameObject list.

### static void saveLevel(GameGrid game, String levelName)

| **Argument** | **Description** |
| --- | --- |
| game | GameGrid to get data from |
| levelName | Name of level to save to |

Iterate through grid writing appropriate character using *getChar* to the file.  
Write other objects to the file using their String representation.

### static char getChar(GameObject go)

Returns: Corresponding character for the GameObject's type.

| **Argument** | **Description** |
| --- | --- |
| go | GameObject with which to find a character |

Defines character representation of types of GameObject. For example, Walls are 'x', Empty space is '.'.  
Must be consistent with *getGameObject*

### static GameObject getGameObject(int x, int y, char c)

Returns: A GameObject of the correct type.

| **Argument** | **Description** |
| --- | --- |
| x | x position in grid |
| y | y position in grid |
| c | the character from the file |

Defines how to choose a GameObject type from a character. For example, 'x' makes a wall, '.' makes empty space.  
Must be consistent with *getChar*

### static String loadText(String filename)

Returns: the file text as a String

| **Argument** | **Description** |
| --- | --- |
| filename | the filename to read from |

Read each line from the file and append it to a String(Builder).

# Class: PauseMenu

public class PauseMenu  
extends MenuContainer

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| Game | game | reference to the current Game object |
| boolean | enabled | is the menu open |
| InvMenuGrid | inv | MenuGrid representing the Game's inventory |

## Methods

### Constructors

#### PauseMenu()

Initialise the container with title of "Pause".  
Adds the pause menu's main options, by creating anonymous class extensions of MenuOption.

### void setGame(Game game)

Set the reference to the Game

| **Argument** | **Description** |
| --- | --- |
| game | Reference to the current Game. |

Also sets InvMenuGrid's reference to the inventory.

# Class: InvMenuGrid

class InvMenuGrid  
extends MenuGrid

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| Map<Item, Integer> | inv | reference to the Game inventory |

## Methods

### Constructors

#### InvMenuGrid(int start\_size)

| **Argument** | **Description** |
| --- | --- |
| start\_size | the starting amount of spaces in the inventory. |

Initialise inventory menu by creating a 5 by 5 grid and adding the start\_size amount of slots.

### setInv(Map<Item, Integer> inventory)

Set the reference to the Game's inventory.

| **Argument** | **Description** |
| --- | --- |
| inventory | inventory reference from the Game |

### void addItem(Item item)

Given an item, add a slot in the inventory.

| **Argument** | **Description** |
| --- | --- |
| item | the Item to add |

If there is an empty slot, add the item to that slot. If there are no empty slots, then add a new one.

# Class: InvSlot

class InvSlot  
extends MenuBox implements MenuAction

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| Item | item | the item that this slot holds |

## Methods

### Constructors

#### InvSlot(Item item)

| **Argument** | **Description** |
| --- | --- |
| item | the item to hold |

Set the item, this object's text to be the same as the item's name and the image to be the item's image.

### void setItem(Item item)

| **Argument** | **Description** |
| --- | --- |
| item | the item to hold |

Set the item, this object's text to be the same as the item's name, and the image to be the item's image.

### void action()

Overridden from super-class.

No action to be performed.

### MenuBox destination()

Returns: the sub-menu to navigate to when selected.

If there is no item, do not navigate. Otherwise navigate to this menu.

# Class: TextboxWriter

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| String | fullText | the whole text to write |
| double | update\_speed | how quickly the text advances. char per frame. |
| double | char\_pointer | the current char to write up to |
| boolean | end | has pointer reached the end of the text |

## Methods

### Constructors

#### Default constructor only

### void setText(String text)

| **Argument** | **Description** |
| --- | --- |
| text | the text to be written |

Set full text, set pointer to start, end is false.

### void update()

Advance the pointer by the amount of update\_speed. If reached the end set end to true.

### void draw(Graphics g)

| **Argument** | **Description** |
| --- | --- |
| g | Graphics object to draw to |

Get the text up to the pointer's position, then split it into lines.  
For each line, write the line's text, then move down.  
Note that there is no wrapping of text, line breaks are only where newline character appears in text.

# Class: MenuContainer

class MenuContainer

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| MenuBox | root | the starting menu item |
| MenuBox | current\_box | the current menu item |
| MenuBox | selection | the currently selected item in the current menu |
| MenuControls | ctrl | keyboard input for controlling in menus |

## Methods

### Constructors

#### MenuContainer(MenuBox root\_menu)

| **Argument** | **Description** |
| --- | --- |
| root\_menu | the menu to set as the root |

Set the root to given object. It becomes the current\_box also. Selection becomes the first child of this object.

### void addChild(MenuBox child)

Add a child to the root.

| **Argument** | **Description** |
| --- | --- |
| child | the item to add to the root menu |

Add the item to the root menu. Set the child's parent reference to be the root.  
If the root is the current menu and selection is unset then the item becomes the selection.

### void navigate(MenuBox to)

Go to the given menu.

| **Argument** | **Description** |
| --- | --- |
| to | The menu to go to |

Set the current active menu to the given item, and set the current selection to be its first child.  
If the given item is null, then don't change.

### void setRootBounds(Rectangle bounds)

| **Argument** | **Description** |
| --- | --- |
| bounds | The dimensions that the menu should take |

Set the bounds of the root menu to the give values.

### void update()

Update the menu for the next frame, processing user input.

Only process input if enough time has passed since last input.  
By default, pressing up goes to previous item, down goes to next.  
For MenuGrids, pressing goes up one row, down goes down, left goes to previous and right goes to next.  
Shift to go up a level, enter to go down a level to the selected item - unless the selected item implements MenuAction then execute its action

### void draw(Graphics g)

Draws the whole menu.

| **Argument** | **Description** |
| --- | --- |
| g | graphics object to draw the menu to |

Draw a translucent rectangle to contain the menu, then call the current menu's draw method.

# Class: MenuBox

public class MenuBox  
implements Comparable

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| MenuBox | parent | the parent of this Box, null if root |
| TreeSet | children | set of children Boxes |
| int | position | ordering among siblings |
| int | depth | how deep into the tree is this |
| String | text | the text to display to represent this |
| Rectangle | bounds | defines space to use when drawing |
| Image | image | the image to represent this when drawing |

## Methods

### Constructors

#### MenuBox(String text, Rectangle bounds)

| **Argument** | **Description** |
| --- | --- |
| text | the text to display to represent this |
| bounds | defines space to use when drawing |

Create new Set for children, initialise members.

### void setBounds(Rectangle bounds)

Set the bounds of the menu. Needed when called to draw so that they are consistent between sub-levels.

| **Argument** | **Description** |
| --- | --- |
| bounds | defines space to use when drawing |

### void addChild(MenuBox menuBox, int position)

Add a child to this menu.

| **Argument** | **Description** |
| --- | --- |
| menuBox | the menu to add |
| position | the position to add it at (Optional) |

The new item's parent is set as this menu, its depth is 1 higher than this.  
If no position is given it can be set to be after all others, otherwise it is as given, then all other positions have to be updated.  
Then add the child to the set.

### void updatePositions(int starting\_position)

Update positions so they are numbered correctly.

| **Argument** | **Description** |
| --- | --- |
| starting\_position | position to start at to skip unnecessary checks |

Start at given position, iterate through remaining members of the set.  
If the next item has the same position as the current, increment it. This fixes positions while allowing non-consecutive positions to be used.

### void setText(String text)

| **Argument** | **Description** |
| --- | --- |
| text | the text to display to represent this |

Set this menu's text.

### int compareTo(Object o)

Overridden from super-class.

One Box is less than another if:  
First its depth is higher (a lower number), then its parent's position is lower, then its position is lower.

### String toString()

Overridden from super-class.

For text-based menu. Result is the this Box's text, then a list of children's text on new lines.

### void draw(Graphics g, MenuBox selected)

Draw the current menu.

| **Argument** | **Description** |
| --- | --- |
| g | the graphics object to draw to |
| selected | the currently selected menu (which should be a child of this) |

Calculate where to draw each line based on the size of the bounding rectangle.  
Write each option on a new line. The currently selected option has an indicator before it.

# Class: MenuAction

public interface MenuAction

## Methods

### void action()

The action to perform when selected

### MenuBox destination()

Returns: where to go when selected

# Class: MenuOption

public abstract class MenuOption  
extends MenuBox implements MenuAction

This class is required because it is not possible to implement an interface in an anonymous class.

## Methods

### Constructors

#### MenuOption(String text)

Call super-class constructor with same arguments.

### void action()

The action to perform when selected

No action to be performed.

### MenuBox destination()

Returns: where to go when selected

Return null. i.e. do not move when this is selected.

# Class: MenuGrid

public class MenuGrid  
extends MenuBox

## Members

| **Type** | **Name** | **Description** |
| --- | --- | --- |
| int | gridWidth | number of columns of the grid |
| int | maxHeight | maximum number of grid rows |

## Methods

### Constructors

#### MenuGrid(String text, Rectangle bounds, int cols, int maxRows)

| **Argument** | **Description** |
| --- | --- |
| text | the text to display to represent this |
| bounds | defines space to use when drawing |
| cols | number of columns of the grid |
| maxRows | maximum number of grid rows (optional) |

Initialise members to given values.

### void draw(Graphics g, MenuBox selected)

Overridden from super-class.

Calculate the size to draw grid squares based on the bounding rectangle.  
Draw each grid space in the correct place, with its image if it has one. If a slot is selected, draw an indicator.