GWD MODIFICATION LANGUAGE

1st Edition

CPSuperstore

Mod Version 0.1.0

To My Sister, Natalie.

Even though you will probably never read this book, but it is still dedicated to you.

Table of Contents

Installing, and Creating a Mod File 6
Using GWD Mod Manager 7
Accessing the Mod Manager 7
Method 1 – Opening In game 7
Method 2 – Opening Externally 7
The Basic Interface 9
Enabling, And Disabling Mods 9
Deleting A Mod 10
Importing Mods 11
Importing Mod Packages 12
Strip Menu Overview 12
Pre-requisites To Creating Mods 13
Metadata Dictionaries 13
Libraries That Can Be Used 14
Commands That Can't Be Used 14
Pygame Blocked Commands 14
Format of this Document 15
Using Command Line Arguments 16
Message Handler Commands 18
Single Message Commands 18
Important Variables 18
Global Values 19
Global Methods 21
Statistics 21
Player Changes 22
Player Properties 22
Player methods 24
Enemy Changes 25
All Enemy Changes 25
All Enemy Properties 25
All Enemy Methods 25

© Copyright CPSuperstore 2018

Single Enemy Changes 27
Single Enemy Properties 27
Single Enemy Methods 28
Star Changes 29
Star Properties 29
Star Methods 29
Bullet Changes 30
All Bullet Changes 30
All Bullet Properties 30
All Bullet Methods 30
Single Bullet Changes 32
Single Bullet Properties 32
Single Bullet Methods 32
Coin Changes 33
All Coin Changes 33
All Coin Properties 33
All Coin Methods 33
Single Coin Changes 34
Single Coin Properties 34
Single Bullet Methods 34
Mystery Box Changes 35
All Mystery Box Changes 35
All Mystery Box Properties 35
All Mystery Box Methods 35
Single Mystery Box Changes 36
Single Mystery Box Properties 36
Single Mystery Box Methods 36
Soup Changes 37
All Soup Changes 37
All Soup Properties 37
All Soup Methods 37
Single Soup Changes 38
Single Soup Properties 38

© Copyright CPSuperstore 2018

Single Soup Methods 38
Black Hole Changes 39
All Soup Changes 39
All Soup Properties 39
All Black Hole Methods 40
Single Black Hole Changes 41
Single Black Hole Properties 41
Single Black Hole Methods 41
Enemy Spawner Changes 42
All Spawner Changes 42
All Spawner Properties 42
All Spawner Methods 42
Single Spawner Changes 43
Single Spawner Properties 43
Single Spawner Methods 43
Custom Objects 44
All Custom Object Changes 44
All Custom Object Properties 44
All Custom Object Methods 44
Single Custom Object Changes 45
Single Custom Object Properties 45
Single Custom Object Methods 46
World Border Changes 47
World Border Properties 47
World Border Methods 48
Audio 49
Sound Objects 49
Channel Map 49
Audio Players 49
Store Control 50
Other Methods 51
Contact 54
Dump Files 54

© Copyright CPSuperstore 2018

Appendix A – Notepad ++ 55
Appendix B – Command List 56

© Copyright CPSuperstore 2018

Installing, and Creating a Mod File

To create a mod, simply navigate to the mods folder, in the same directory as *Galaxy Wide Domination.exe*. If the folder does not exist, create one, with the name *mods*. You must be inside this directory to continue. (if you installed through Steam, by default it should be *C:\Program Files (x86)\Steam\steamapps\common\GalaxyWideDomanation*)

Now that you are in the mods directory, to install a mod, simply paste the file, and all other dependencies here.

When developing a mod, use this folder to work, and store all mod dependencies here as well. The game will only recognize the file extension *.gwd*. This means that the *.gwd* file MUST be the main file, however, you can include .py files and import those just fine. Please note that each *.gwd* file will be treated as an individual mod. (This different extension was chosen to eliminate confusion between different mods, and file dependencies)

All mods in this folder will each be compiled. If one mod fails, the others will still run. If a mod fails, a red message will be displayed in the console, and a dump file will be created in the same directory as the mods. It will be called *[Mod Name]_dump_[Unix time].dump*. It will store the full Stack Traceback of the file.

The syntax for this language is Python. Any python import/statement should work (see library section for a list of available libraries).

To use a mod once installed, simply launch the game, and select the *mods* option on the main menu. This will run the default game mode, but with all the mods running. While loading, depending on the size and number of mods, this process could take a while.

Please note that progress in a modded game WILL NOT be saved. Using a modded game will disable the story, and not allow your score to be sent to the high scores server. If you play on a modded server, the mods will automatically be modified to fit the server's needs, with all old mods being stored in a different sub-folder called *old*.

DISCLAIMER CPSuperstore Inc cannot be held responsible for damage, loss or any other event due to a malicious mod, be it from directly installing, or automatically installed from a server. You were warned.

DISCLAIMER CPSuperstore Inc holds the right to take any mod created by anyone and add it as a full feature to the game, with or without explicit consent from the original author, however additional assets such as images and sounds remain sole property of the original author.

© Copyright CPSuperstore 2018

Using GWD Mod Manager

Accessing the Mod Manager

Method 1 – Opening In game

To access the GWD mod manager, start by launching the game.



Figure 1-1. The game's main menu.

From the menu, use the arrow keys to hover over the "Mods" option, and press "Enter" to select it. This will bring up the mod options screen:



Figure 1-2. The Mod Options Screen

Once you have reached this screen, simply press the "M" button on your keyboard. This will launch the mod manager. To resume the game, simply close the manager. All changes will take effect instantly.

Method 2 – Opening Externally

First, you must navigate to the GWD installation directory. If you do not remember where this is, open the game properties in Steam. This is done by right-clicking the game in your Steam library, and clicking "Properties"

© Copyright CPSuperstore 2018

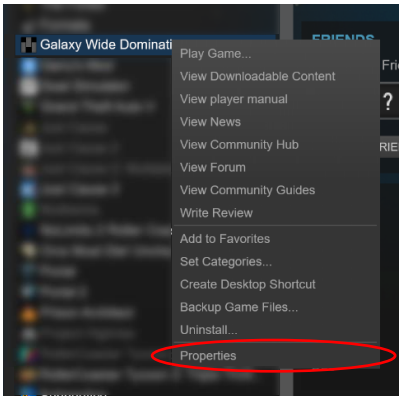


Figure 1-3. Right-Clicking the game in the Steam library.

This will open the game properties window. Select the “Local Files” tab, and clicking the “Browse Local Files” button.

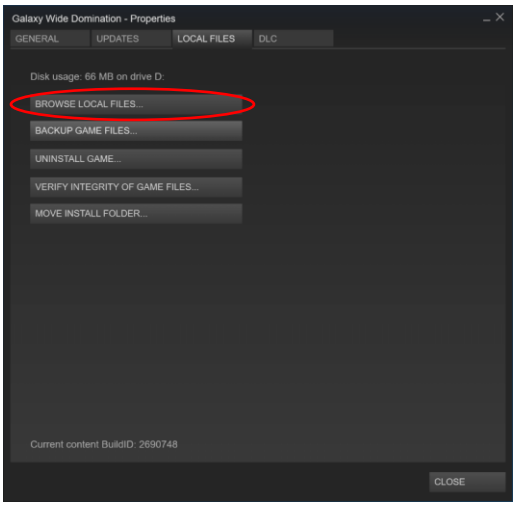


Figure 1-4. The Game’s Local Files Properties

This will bring you to the installation directory. From here, simply locate, and double-click “ModManager.exe”.

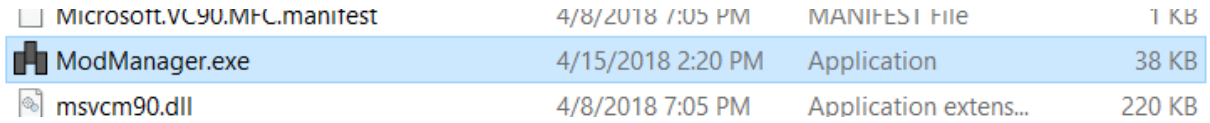


Figure 1-5. The Manager in Explorer

This will launch the mod manager. The changes will take affect instantly. If you are playing in modded mode while managing the mods, you will need to leave the current game, and go back to see the changes.

© Copyright CPSuperstore 2018

The Basic Interface

Mod management has just become a walk in the park, thanks to the new *Mod Manager*! This program allows you to enable, disable, import, and delete GWD mods with a very simple user interface.

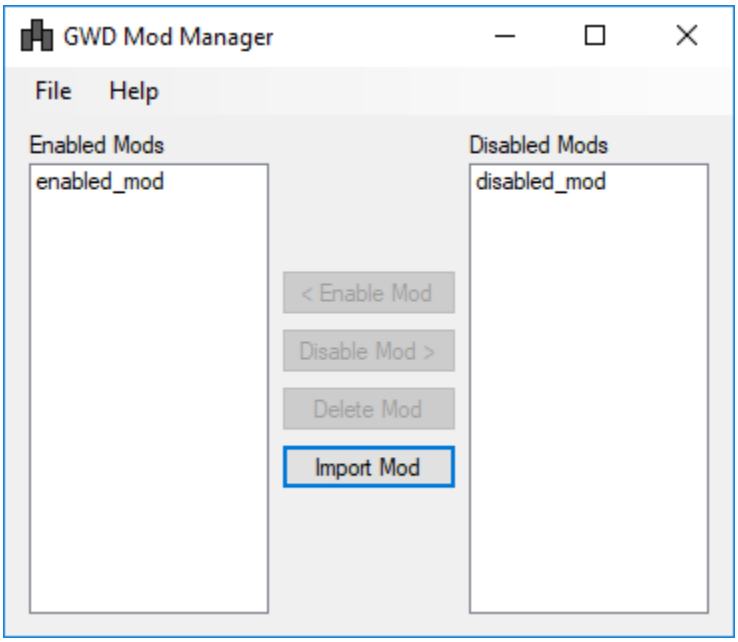


Figure 2-1. The Main Program Interface.

In this example, there are two mods. One called “enabled_mod”, and another called “disabled_mod”.

Enabling, And Disabling Mods

To enable a mod, simply click on it in the “Disabled Mods” list.

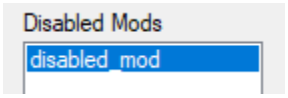


Figure 2-2. A Selected Mod.

Once the mod is selected, the “Enable Mod” Button will become available. Simply click the button to enable the mod.

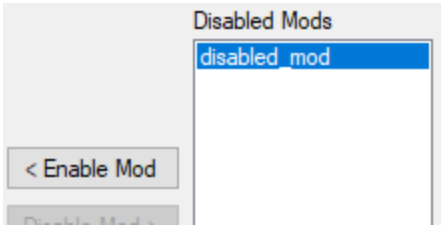


Figure 2-3. The “Enabled Mod” button.

© Copyright CPSuperstore 2018

The mod name will be moved over to the “Enabled Mods” List. In order to disable a mod, simply click on the mod in the “Enabled Mods” list and click the newly available “Disable Mod” button.

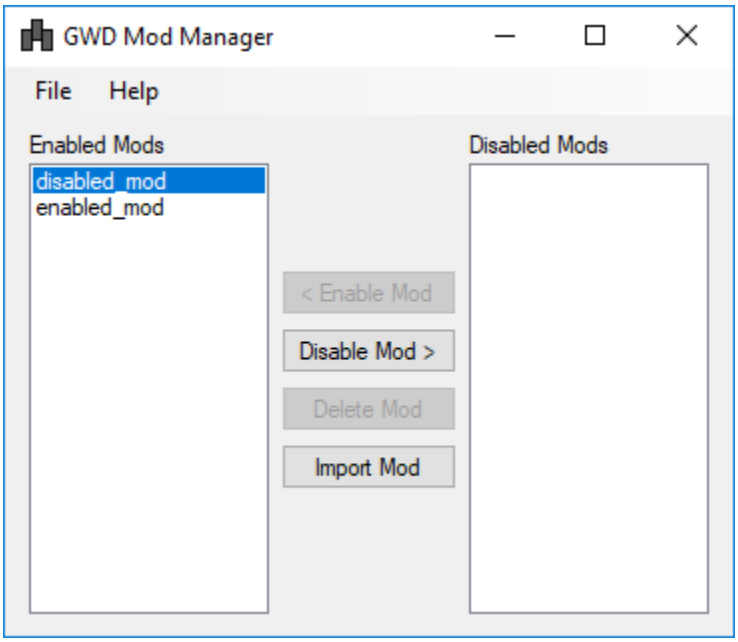


Figure 2-4. Both Mods Are Enabled.

Deleting A Mod

When you delete a mod, the mod is deleted forever, and can not be recovered. To delete a mod, it must first be disabled (See Enabling and Disabling Mods).

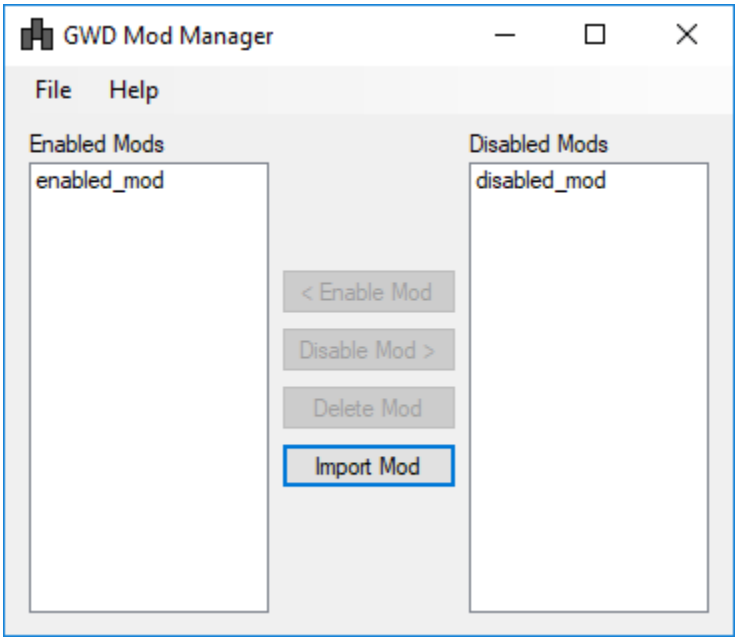


Figure 2-5. The Interface with a Disabled Mod

© Copyright CPSuperstore 2018

Now that the mod is disabled, click the newly Enabled “Delete Mod” Button.

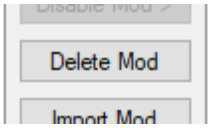


Figure 2-6. The “Delete Mod” Button

Once you click this button, a conformation dialog box will open. It will ask you if you are sure you want to delete the mod. Press “Ok” To delete the mod, or “Cancel” to cancel the operation. PLEASE NOTE: THIS ACTION CAN NOT BE UNDONE!

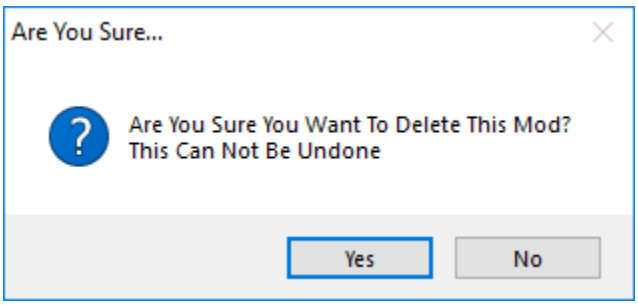


Figure 2-5. The Deletion Confirmation Dialog Box.

Importing Mods

To import a mod from your computer, simply click the “Import Mod” button.

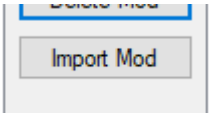


Figure 2-6. The “Import Mod” Button.

Once this button is clicked, a file dialog will open. This will allow you to select a file with the extension “.gwd”. This is the only accepted file extension for mods. Select a file, and click “Open”. This will add the mod to the list of “Enabled” Mods.

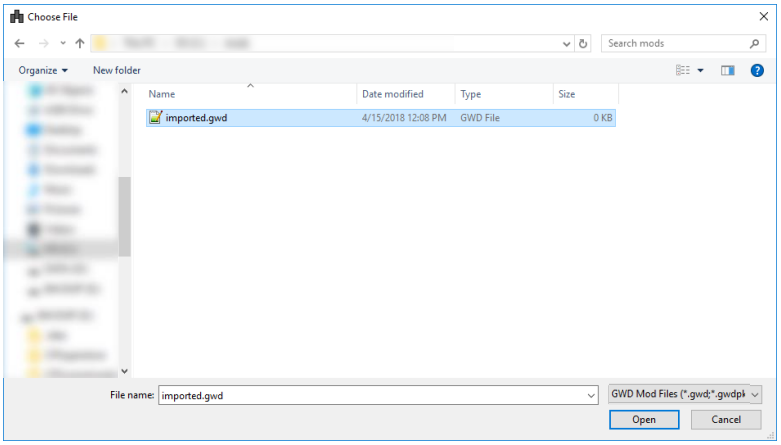


Figure 2-7. The File Explorer. Please Note, Additional Blurring Was Used for Privacy Reasons.

© Copyright CPSuperstore 2018

Importing Mod Packages

Often, a mod will need to be larger than a single file. It may contain additional image, sound, or other files. To prevent confusion with mod installation, a new mod file type was created. It is the *GWD Mod Package*. This file uses the file extension *.gwdpkg*. When this type of file is imported, the *Mod Manager* will deal with the file dependencies, and placing all of the additional files where they need to go. Importing a package can take a while, depending on the number of files it must deal with. Installing mod packages is basically the same process as a regular mod.

Start by clicking the "Import Mod" button.

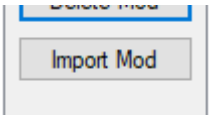


Figure 2-8. The "Import Mod" Button.

Once this button is clicked, a file dialog will open. This will allow you to select a file with the extension ".gwd", or ".gwdpkg". These are the only accepted file extensions for mods. Select a ".gwd" file, and click "Open". Wait a few seconds, and the mod will be imported!

Strip Menu Overview

In the strip menu (the menu at the top of the window) there 2 sections available are *File*, and *Help*. The *File* section contains program options. The *Help* section contains help and support information about the program. Inside the *File* section, you will find two subsections, *Reload Mods*, and *Exit*.

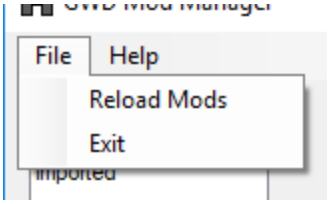


Figure 2-8. The *File* Menu.

The *Reload Mods* option will make updates to the mods in the program. This is useful if there was a manual change to the mods folder. The *Exit* option simply confirms and closes the program.

The *Help* menu also contains two options. *Help*, and *Version Info* (Figure 2-3)

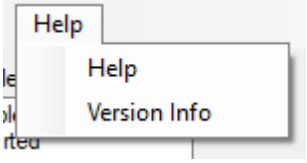


Figure 2-9. The *Help* Menu

© Copyright CPSuperstore 2018

The *Help* option displays a dialog box which shows a description of the program, and the email address to contact in the event of an issue (cpsuperstoreinc@gmail.com).

The *Version Info* option displays the version number, build, code name, and date of the build you are using. This information is important if you are experiencing an issue with the program, and you contact support. This will help the support team better understand what went wrong.

Pre-requisites To Creating Mods

The language the mod will be written in is Python (2.7). The graphics library is Pygame. To write a mod of your own, you should be familiar with basic Python 2.7 commands, and conventions. Mods do not require any special installation, and there is no IDE available to do development in at the time of this writing. I would recommend the use of *Notepad++* (https://notepad-plus-plus.org/) for writing mods, however, it does not matter.

Metadata Dictionaries

Metadata dictionaries are dictionaries that should, but do not have to be placed at the beginning of the file. They provide additional information on the mod file. The Game Modifier mods will use this information to display should the user attempt to use a modifier. They are also a great way to provide additional information to another mod developer, or to yourself if you leave a project and come back after a long period of time.

Metadata dictionaries are surrounded by curly brackets and can span over multiple lines. The recommended fields to contain in a docstring are the name of the name of the mod, the mod's author, the date the mod was last modified, the mod's version, the author's primary contact information (in the event a user encounters an issue with the mod), and a description of what the mod is supposed to do.

```
{
    "name": "Hardcore",
    "author": "CPSuperstore",
    "date": "7/24/2018",
    "version": "0.1.0",
    "contact": "cpsuperstoreinc@gmail.com",
    "description": "No Soup. No Healing. 1 Life. Good Luck.",
}
```

Listing 3-2. An example of a metadata dictionary found in the "Hardcore" game modifier (game_modifiers/hardcore.gwd)

Libraries That Can Be Used

If a library is imported that is not part of this list (including other files), the mod will be flagged as malicious, and WILL NOT compile. For more information or would like a module to be added to this list, feel free to contact me at cpsuperstoreinc@gmail.com, and I will get back to you as soon as possible. To avoid confusion, please only import one module per line. If not, the system will detect the module as malicious, and will not compile.

This system has been put in place to prevent users from doing harm to the computers of the mod users. The selected modules are unable to do any damage to the user's computer.

1. random – allows generation of random figures in a specified range.
2. TkInter – allows for form-based GUIs.
3. pygame – allows for GUI applications. The game was built in pygame.
4. Pygame.locals – pygame extension for interacting with keyboard events
5. time – allows for time delays and tracking
6. math – allows for more advanced mathematical operations
7. threading – allows for multiple processes to be run at once

Commands That Can't Be Used

The following list of commands have been disabled as they cause interference with the actual game. For more information, or would like a module to be added to this list, feel free to contact me at cpsuperstoreinc@gmail.com, and I will get back to you as soon as possible. To avoid interference, the mod compiler will replace any line with the following commands in it, with a blank line, and still attempt to run the mod. This will almost always result in a failure, if the mod is dependent on one of these lines..

Pygame Blocked Commands

1. display.set_mode – mods cannot create pygame displays. There is already one, and it is called *screen*

© Copyright CPSuperstore 2018

Format of this Document

For the list of commands, the format is as follows

function_name():
    description
    `function_name_with_arguement_names_and_types()`
    the type and name of the function's return value

For specific properties, the format will be
property_name
    description
    expected_type

PLEASE NOTE: If a function requires an argument called *screen*, always pass in the word *screen*. This is the pygame surface object the entire game is built on. By passing in some other value, the mod is guaranteed to crash.

© Copyright CPSuperstore 2018

Using Command Line Arguments

To speed up mod development, we added support for command-line arguments to start the game in any native game mode. To start the game with a command-line argument, on Windows, simply navigate to the same directory as the main program executable (this can be found my using the same method as locating the mod manager).

Once in the directory, simply make sure no files are selected. Then hold down the *SHIFT* button, and *RIGHT-CLICK* anywhere which is NOT on a file. In the context menu, select "Open PowerShell Window Here". The text may also be "Open Command Prompt Here" depending on your version of Windows.

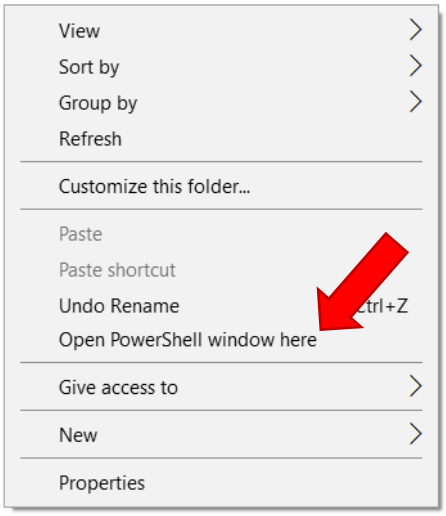


Image 7-1. The Special Context Menu.

This will open the Windows Bash Shell. It is extremely important you only issue commands that you are familiar with, as you can do a lot of damage to your computer with the wrong command.

If you have opened Windows PowerShell, or you are unsure which one (PowerShell, or Command Prompt) you are using, simply issue the following command

```
cmd
```

Command 7-1. The command to open Command Prompt from PowerShell

To run the game normally, simply issue the following command:

```
"Galaxy Wide Domination.exe"
```

Command 7-2. The command to open the game normally.

If you created a batch file, you could also use this command to run the game. If you decide to store the batch file in another directory, remember to include the path to the directory between the first quote, and the "G" in "Galaxy". Please Note: Creating Batch files is outside the scope of this book.

© Copyright CPSuperstore 2018

To start the game in any of the other game modes, simply add the name of the mode after the second double quote

```
"Galaxy Wide Domination.exe" [modeName]
```

Command 7-3. The command to open the game in the specified mode.

In the above command, simply replace the `[modeName]` placeholder in the command with one of the following modes:

Mode Keyword	Description
`normalStart`	This keyword will start the game normally
`noLauncher`	This keyword will start the game, but without showing the launcher
`singlePlayer`	This keyword will start the game in single player story mode
`modded`	This keyword will start the game in modded mode
`tutorial`	This keyword will start the tutorial, and after that, it will start the menu
`demo`	This keyword will start the game in single player mode, but with the demo restrictions enabled

After the game finishes which ever mode was specified, it will run as if it had been launched normally.

Here is an example using these Start Keywords. In this example, the game will be started in modded mode.

```
"Galaxy Wide Domination.exe" modded
```

Command 7-4. The command to open the game in the specified mode.

© Copyright CPSuperstore 2018

Message Handler Commands

The message handler is the object that oversees handling all messages and global message properties.

To access the pre-existing message object with the `message` keyword.

message.add_message()

Displays the text to the console

`add_message(string message)`

returns the individual message object

message.clear_message()

Clears the entire console of ALL messages

`clear_message()`

returns nothing

Single Message Commands

The single message is the object of a single message. This is just one line of output in the console

This object is created using the `message.add_message()` command.

messageObject.set_text()

Changes the text of a message in the console

`set_text(string message)`

returns nothing

messageObject.delete()

deletes the message from the console

`delete()`

returns nothing

Important Variables

The mod allows access to some important variables. These variables are listed in a table format

Name	Type	Description
`screen`	Pygame Display / Surface	The main display/screen object

© Copyright CPSuperstore 2018

Global Values

These values are the values that control most of the game. Of all of the properties, these ones are the most powerful.

GlobalVariable.LITE_VERSION
: True if the game is running in LITE mode
Boolean

GlobalVariable.coins
: the number of coins the player has
int

GlobalVariable.game_level
: The current game level the player is at. Changing this will not change the level of the game until the player has leveled up
int

GlobalVariable.nuke_load
: The level the nuke load is at. This number by default is out of 300. This loads one number per frame
int

GlobalVariable.force_shield
: If the force shield is enabled in the player
Boolean

GlobalVariable.score
: the current score of the player. If mods are enabled (which they will automatically be if modded mode is enabled)
int

GlobalVariable.derp_mode
: If the color flashing mode (most commonly caused by acquiring soup) is enabled
Boolean

GlobalVariable.derp_mode_dur
: The length of time in seconds that derp mode will last.
int

GlobalVariable.ghost_speed
: The speed the player can travel at if they are in ghost (spectator) mode. This is measured in pixels moved per frame
int

© Copyright CPSuperstore 2018

GlobalVariable.framerate

The framerate of the game. It is the average of the last ten frames. For the first ten frames, this will be 0.
float

GlobalVariable.coin_reminder

the number of coins the player has to receive before the game reminds the player to visit the store. (Ex, if 20, every time the player gets 20 coins, they receive a reminder)
int

GlobalVariable.FONT

the path to the main game font file (can be *.ttf, .otf,* or the name of an installed font)
string

GlobalVariable.ALT_FONT

the path to the alternative game font file (can be *.ttf, .otf,* or the name of an installed font)
string

GlobalVariable.CREDIT_FONT

the path to the credits font file (can be *.ttf, .otf,* or the name of an installed font)
string

GlobalVariable.statistics

A dictionary of all statistics from the user. See the *Statistics* section of this document
dict

GlobalVariable.ghost_ship_probability

The probability of an enemy spawning as a ghost ship. For example, if the probability should be 1 out of 100 (1 / 100 – 1%), set this value to 100.

GlobalVariable.coin_drop_chance

The percent chance of an enemy dropping a coin on death. For example, if a coin should always be dropped, this should be set to 100, but if half of the time is desired, set the value to 50.

© Copyright CPSuperstore 2018

Global Methods

GlobalVariable.set_time_to_play()

Sets the total time the user can play the game in seconds

`set_time_to_play(int seconds)`

returns nothing

GlobalVariable.get_time_left()

Returns the number of seconds left in the game, or `None` if the total time has not been set

`get_time_left()`

returns seconds left in game or `None`

Statistics

These values are tracked by the game. They can also be set by a mod.

GlobalVariable.statistics[“shots_fired”]

The total number of times the player has shot a bullet per game

int

GlobalVariable.statistics[“enemy_hits”]

The total number of times the player has shot an enemy per game

int

GlobalVariable.statistics[“enemy_kills”]

The total number of times the player has killed an enemy per game

int

GlobalVariable.statistics[“times_shot”]

The total number of times the player been shot by an enemy per game

int

GlobalVariable.statistics[“times_killed”]

The total number of times the player has been killed per game

int

GlobalVariable.statistics[“coins_spent”]

The total number of coins the player has spent per game

int

GlobalVariable.statistics[“coins_collected”]

The total number of coins the player has collected per game

int

© Copyright CPSuperstore 2018

Player Changes

The player is the green ship in the center of the screen. It is the object that the player can control.

Player Properties

player.hp

The amount of health the player currently has
int

player.max_hp

The maximum amount of health the player can have
int

player.heal_rate

The amount of health the player will receive per second
int

player.level

The current ship level the player has. Changing this will not upgrade the ship. Please refer to `player.set_level()` in the *Player Methods* subsection of the *Player Changes* section
int

player.name

The name of the player. Please refrain from changing this value, however, if is changed, and the player exits out of the game, the name will be set to the user specified name
string

player.x

The x position of the player relative to the screen (not the world)
int

player.y

The y position of the player relative to the screen (not the world)
int

player.w

The width of the player's image
int

player.h

the height of the player's image
int

© Copyright CPSuperstore 2018

player.angle

The angle the user is facing, where 0° / 360° is facing up, and as the angle increases, the player will turn left
int

player.original_angle

The angle the player will be set to on death or completion of a level
int

player.speed

the speed the player can fly at in pixels per second
int

player.fire_rate

The inverse of the number of bullets that can be fired per second
(Example, If the player should shoot three bullets per second, set this value to 1/3)
float

player.lives

The remaining number of lives the player has
int

player.bullet_level

the level (and radius in pixels) of the player's bullet level. The bullet level multiplied by two will be the amount of damage dealt on impact. Changing this value will take effect immediately
int

© Copyright CPSuperstore 2018

Player methods

player.reset_player()

Resets the player's health and angle to the default values

```
reset_player()
```

returns nothing

player.add_speed()

Adds (or subtracts) speed from the player by amount (use negative number to subtract).

```
add_speed(int amount)
```

returns nothing

player.add_bullet_level()

Adds (or subtracts) the bullet level from the player by amount
(use negative number to subtract).

```
add_bullet_level(int amount)
```

returns nothing

player.heal()

Entirely heals the player (sets `player.hp` to `player.max_hp`)

```
rotate(int direction)
```

returns nothing

player.rotate()

Rotates the player by a specified amount of degrees (positive for left, negative for right)

```
clear_message(int direction)
```

returns nothing

player.get_real_pos()

returns the players x, and y position relative to the screen (not the world)

```
get_real_pos()
```

returns tuple of 2 integers which represent the player's x, and y position respectively

player.get_fire_spot()

returns the x, and y position of where the player's bullets will originate from

```
get_fire_spot()
```

returns tuple of 2 integers which represent the player's fire position respectively

player.get_rect ()

returns a pygame rectangle (rect) object of the player

```
get_rect()
```

returns the pygame rect object of the player

© Copyright CPSuperstore 2018

player.set_level()

Sets the player's upgrade level. Setting this value will update the player immediately

```
set_level(int level)
```

returns nothing

player.get_health()

returns the player's current health

```
get_health()
```

returns the player's current health as a percent

Enemy Changes

The enemy ships are the red ships that the player must fight in the game.

All Enemy Changes

These are the changes that affect all of the enemies in the game as a whole object.

All Enemy Properties

bad.turn_magnatude

The maximum change in angle the enemy could take per frame (default = 5) in either direction

int

bad.bad_dudes

The list of all enemies currently alive in the game

list

All Enemy Methods

bad.make_point_list ()

returns a list of the coordinates of each enemy in the system

```
make_point_list()
```

returns the list of coordinates

bad.add_enemies()

Adds a specified number of enemies to the level. Each will share a common image, but other properties will be generated based from the player's upgrades

```
add_enemies(int amount, str image_path)
```

returns nothing

bad.move()

Moves all enemies in the system by x, and y

```
move(int x, int y)
```

returns nothing

© Copyright CPSuperstore 2018

bad.get_number_of_enemies()

Returns the total number of enemies in the system still alive

```
get_number_of_enemies()
```

returns number of alive enemies

bad.add_custom_enemies()

Adds a bad guy to the level based on the parameter specifictions

```
add_custom_enemy(
      int x, int y, image, hp=50, heal_rate=0, level=1, name="",
      speed=3, fire_rate=5, bullet_level=1
)
```

returns the object of the enemy created

bad.add_custom_enemy_range()

Adds a bad guy to the level based on the parameter specifications, which could be spawned anywhere in the world

```
add_custom_enemy_range(
      image, hp=50, heal_rate=0, level=1, name="", speed=3,
      fire_rate=5, bullet_level=1
)
```

returns the object of the enemy created

bad.is_point_on()

Returns ONE enemy which is touching a specified point

```
is_point_on(tuple point)
```

returns ONE enemy which is toughing the point

bad.is_alive()

Returns if the specified enemy is currently alive

```
is_alive(enemy single_enemy)
```

returns if the enemy is alive

bad.get_random_enemy()

Returns one random enemy from all of the alive enemies

```
get_random_enemy()
```

returns a random enemy which is currently alive

© Copyright CPSuperstore 2018

Single Enemy Changes

These changes affect a single specified enemy. If the selected enemy dies, the updates will no longer work.

Single Enemy Properties

enemyObject.image
: the image of the enemy
: string

enemyObject.hp
: the enemy's health
: int

enemyObject.heal_rate
: the rate at which the player will heal in health gained per second
: int

enemyObject.level
: the upgrade level of the enemy
: int

enemyObject.name
: the name of the enemy (it is displayed underneath the player)
: string

enemyObject.x
: the enemy's x position relative to the screen
: int

enemyObject.y
: the enemy's y position relative to the screen
: int

enemyObject.angle
: the enemy's current angle
: int

enemyObject.speed
: the speed at which the enemy will move at in pixels per frame
: int

© Copyright CPSuperstore 2018

enemyObject.fire_rate

the inverse of the number of bullets that can be fired per second
(Example, if the enemy should fire thee bullets per second, set this value to 1/3)
float

enemyObject.bullet_level

the bullet level of the enemy
int

enemyObject.natural_turn

if the enemy should continuously change angle while flying
boolean

Single Enemy Methods

enemyObject.get_real_pos()

Returns the x, and y position of the enemy relative to the screen
`get_real_pos()`
returns a tuple of the x and y position of the enemy

enemyObject.get_fire_spot()

Returns the x, and y position of where the enemy's bullet should originate from
`get_fire_spot()`
returns a tuple of the x and y position of the enemy

enemyObject.get_rect()

Returns the pygame rectangle (rect) object of the player's image
`get_rect()`
returns the pygame rect object of the player

© Copyright CPSuperstore 2018

Star Changes

The stars are simply the randomly placed, and sized circles in the background. They serve no real purpose, aside of enforcing the illusion of movement.

Star Properties

star.star_array

This is the array which stores all the stars in the map.

list

Star Methods

star.get_rect()

Adds a new star (a colored circle) to a random location on the map

```
add_star()
```

returns nothing

star.move()

Moves all of the stars in the system by x, and y pixels

```
move(int x, int y)
```

returns nothing

© Copyright CPSuperstore 2018

Bullet Changes

Bullets are the objects that the enemies, and player shoots at each other.

All Bullet Changes

These changes affect all the bullets globally.

All Bullet Properties

bullet.bullets

This is the array which stores all the bullets in the map.
list

bullet.bullet_damage_multiplier

This number is multiplied by the bullet's level (radius) to calculate toe total damage the bullet will cause on impact.
int

All Bullet Methods

bullet.add_bullet()

Adds a bullet to the game. The position, and angle are automatically determined from the passed in object reference
(Example, if the player should shoot, please use *player*, and not *'player'*)

```
add_bullet(object fired_by)
```

returns nothing

bullet.move()

Moves all of the bullets in the system by x, and y pixels

```
move(int x, int y)
```

returns nothing

bullet.nuke ()

Sends a nuke from the specified object.
(Please note, at the time of this writing, the nuke will only kill enemy ships. including an enemy ship that has fired the nuke)
(Example, if the player should launch, please use *player*, and not *'player'*)

```
nuke(object entity)
```

returns nothing

© Copyright CPSuperstore 2018

bullet.is_collided_with()

checks if any bullet has collided with the following specifaications

(If x, and y are not specified, the position of the entity will be used)

```
is_collided_with(
      object object, int x=None, int y=None,
      bool enemy_friendly_fire=True, bool get_who_shot=False
      )
```

returns the anount of damage to deal from the object. If *get_who_shot* is set to *True*, a list will be returned containing the damage as item 0, and the object which shot the bullet will be item 1.

bullet.clear()

Clears all of the bullets in the system.

```
clear()
```

returns nothing

© Copyright CPSuperstore 2018

Single Bullet Changes

These changes affect only a single specified bullet.

Single Bullet Properties

bulletObject.fired_by

This is the entity who shot the bullet. The entity can shoot any other entity except for its self (unless otherwise specified).
object

bulletObject.radius

This is the radius of the bullet. The total damage dealt on impact with this bullet is calculated by using the following formula:
$damage = radius \times bullet_damage_multiplier$
int

Single Bullet Methods

bulletObject.get_real_pos ()

returns the position of the bullet relative to the screen.
`get_real_pos()`
returns a tuple of the x, and y position

bulletObject.get_rect ()

returns the pygame rect object of the bullet.
`get_rect()`
returns pygame rect object

bulletObject.in_force_shield ()

determines if the bullet is inside the force shield, and if the shield is active
`in_force_shield()`
returns if the bullet is in the active force shield.

© Copyright CPSuperstore 2018

Coin Changes

The coins are the object the enemy drops on death. When the player picks up the coin, they can spend it in the store.

All Coin Changes

These changes affect all the coins as a whole.

All Coin Properties

coin.coin_array

This is the entire list of all singular coins in the system.

list

All Coin Methods

coin.add_coin()

adds a new coin at the specified position, relative to the screen

```
add_coin(int x, int y)
```

returns the newly created coin

coin.move ()

moves each coin in the system by a factor of *x*, and *y*

```
move(int x, int y)
```

returns nothing

coin.is_touching()

detects if any coin is touching the specified object.

```
is_touching(object gameObject)
```

returns if the coin is touching as a boolean value

coin.get_coin_list()

returns a list of the coordinates of each coin in the game.

```
get_coin_list()
```

returns a list of the coordinates of the coins

coin.clear_coins()

clears all of the dropped coins in the map.

```
clear_coins()
```

returns nothing

coin.get_count()

returns the number of coins in the level.

```
get_count()
```

returns the number of coins in the level

© Copyright CPSuperstore 2018

Single Coin Changes

These changes affect a specified single coin object.

Single Coin Properties

coinObject.x

the x position of the coin relative to the screen

int

coinObject.y

the y position of the coin relative to the screen

int

coinObject.w

the width of the coin object (image)

int

coinObject.h

the width of the coin object (image)

int

coinObject.value

the number of money the player will receive when in contact with the coin

int

Single Bullet Methods

coinObject.get_rect()

generates the pygame rectangle (rect) object of the coin.

`get_rect()`

returns a pygame rect object

© Copyright CPSuperstore 2018

Mystery Box Changes

The mystery box is the purple box that flies across the screen when the player completes a game level. On contact, a mystery action will execute, with 50% chance of the item being a helpful item.

All Mystery Box Changes

These changes affect each mystery box in the game.

All Mystery Box Properties

box.w

the width of each box object (image)
int

box.h

the height of each box object (image)
int

box.boxes

the list of each mystery box in the system
list

All Mystery Box Methods

box.add_box()

adds a new mystery box to fly across the screen
`add_box()`
returns the new box object

box.move()

moves all of the mystery boxes by a factor of *x*, and *y*
`move(int x, int y)`
returns nothing

box.is_touching()

detects if an object is touching any of the mystery boxes.
`is_touching(object gameObject)`
returns a boolean if the detection exists

box.get_box_list ()

generates a list of the coordinates of each box in the game.
`get_box_list()`
returns a list of coordinates

© Copyright CPSuperstore 2018

Single Mystery Box Changes

These changes affect an individual mystery box.

Single Mystery Box Properties

boxObject.x

the x position of the box relative to the screen
int

boxObject.y

the y position of the box relative to the screen
int

boxObject.angle

the angle the box will fly at, where 0° is flying to the left, and as the angle increases, the box will rotate counter-clockwise
int

boxObject.speed

The speed the box will move at (in pixels per frame)
int

boxObject.image

the path to the box's image
string

Single Mystery Box Methods

boxObject.get_rect()

generates a pygame rectangle (rect) object of the mystery box
`get_rect()`
returns a pygame rect object

© Copyright CPSuperstore 2018

Soup Changes

The soup is the can which changes color as it flies across the screen on level completion. When the user contacts the soup, the player will lose two coins, but will receive full health.

All Soup Changes

These changes affect one single can of soup

All Soup Properties

soup.w

the width of each soup object (image)
int

soup.h

the height of each soup object (image)
int

soup.soup

the list of each mystery soup in the system
list

soup.spawn_soup

if the game should naturally spawn soup
Boolean

All Soup Methods

soup.add_soup()

adds a new mystery soup to fly across the screen
`add_soup()`
returns the new soup object

soup.move()

moves all of the mystery soupes by a factor of *x*, and *y*
`move(int x, int y)`
returns nothing

soup.is_touching()

detects if an object is touching any of the mystery soupes.
`is_touching(object gameObject)`
returns a boolean if the detection exists

soup.get_soup_list ()

generates a list of the coordinates of each soup in the game.
`get_soup_list()`
returns a list of coordinates

© Copyright CPSuperstore 2018

Single Soup Changes

The following changes pertain to individual cans of soup

Single Soup Properties

soupObject.x

the x position of the soup relative to the screen
int

soupObject.y

the y position of the soup relative to the screen
int

soupObject.angle

the angle the soup will fly at, where 0° is flying to the left, and as the angle increases, the soup will rotate counter-clockwise
int

soupObject.speed

The speed the soup will move at (in pixels per frame)
int

Single Soup Methods

soupObject.get_rect()

generates a pygame rectangle (rect) object of the mystery soup
`get_rect()`
returns a pygame rect object

© Copyright CPSuperstore 2018

Black Hole Changes

The Black Holes spawn randomly throughout the game. They suck everything in, including you, enemies, and even bullets.

All Soup Changes

These changes affect one single can of soup

All Soup Properties

black_hole.black_hole_array
: the list of all alive black holes
: list

black_hole.spawn_delay_min
: the minimum amount of time between black holes spawning. The default is 50 seconds.
: int

black_hole.spawn_delay_max
: the maximum amount of time between black holes spawning. The default is 100 seconds.
: int

black_hole.lifetime_min
: the minimum amount a natural black hole can be alive. The default is 10 seconds.
: int

black_hole.lifetime_max
: the maximum amount a natural black hole can be alive. The default is 20 seconds.
: int

black_hole.natural_black_holes
: True if black holes can spawn naturally
: Boolean

© Copyright CPSuperstore 2018

All Black Hole Methods

black_hole.add_black_hole()

adds a new black hole to spawn at the specified cordimates.

```
add_black_hole(int x, int y, int lifetime=random)
```

returns the new black hole object

black_hole.move()

moves all of the black holes by a factor of *x*, and *y*

```
move(int x, int y)
```

returns nothing

black_hole.is_touching()

detects if an object is touching any of the black holes.

```
is_touching(object gameObject)
```

returns a boolean if the detection exists

black_hole.get_black_hole_list ()

generates a list of the coordinates of each black hole in the game.

```
get_black_hole_list()
```

returns a list of coordinates

© Copyright CPSuperstore 2018

Single Black Hole Changes

The following changes pertain to individual black holes

Single Black Hole Properties

blackHoleObject.x

the x position of the black hole relative to the screen
int

blackHoleObject.y

the y position of the black hole relative to the screen
int

blackHoleObject.w

the width of the black hole image
int

blackHoleObject.h

the height of the black hole image
int

blackHoleObject.expiration

the UNIX time when the black hole will die
float

Single Black Hole Methods

soupObject.get_rect()

generates a pygame rectangle (rect) object of the mystery soup
`get_rect()`
returns a pygame rect object

© Copyright CPSuperstore 2018

Enemy Spawner Changes

The enemy spawner is the blue and red object which is constantly rotating. By default, this will spawn an enemy once every ten seconds.

All Spawner Changes

These changes affect all of the spawners

All Spawner Properties

spawn_handler.enemy_handler

The object where the spawned enemies will be stored
EnemyHandler

spawn_handler.spawners

The list of spawners which are active
list

All Spawner Methods

spawn_handler.update()

Updates each spawner (this will draw the spawners, and add the enemy if enough time has passed)
`update()`
returns nothing

spawn_handler.move()

This function moves each spawner by a factor of x, and y
`update(int x, int y)`
returns nothing

spawn_handler.add_spawner()

Adds a new spawner to the game
`update(int x, int y, list objects_to_spawn, int spawnrate=10)`
returns nothing

spawn_handler.get_spawner_coords()

returns a list of all coordinates of all of the spawners in the game.
`get_spawner_coords()`
returns list

© Copyright CPSuperstore 2018

Single Spawner Changes

Single Spawner Properties

spawnerObject.x_pos

the x position of the spawner
int

spawnerObject.y_pos

the y position of the spawner
int

spawnerObject.spawn_objects

the list of objects that could be spawned (a list of enemies)
list

spawnerObject.bad

the object which handles all of the spawned enemies
enemyHandler

spawnerObject.last_spawn

the Unix time of the last spawn
float

spawnerObject.spawn_delay

the amount of time between enemies spawning (in seconds)
int

spawnerObject.angle

the angle which the spawner is facing (between 0 and 360 degrees)
int

Single Spawner Methods

spawnerObject.update()

Updates the spawner (this will draw the spawner, and add the enemy if enough time has passed)
`update()`
returns nothing

© Copyright CPSuperstore 2018

Custom Objects

Custom objects are not in the un-modded game. They are here to simply allow mod creators to develop their own objects and put them in the game.

All Custom Object Changes

The following objects effect every custom object in the game

All Custom Object Properties

custom.objects

The list of each custom object in the system

list

All Custom Object Methods

custom.add_object()

adds a new custom object to the system

```
get_object(
      int x, int y, string image, string name="", int angle=0, boolean
      ai=False, int speed=1
)
```

returns the custom object

custom.is_touching()

determines if the specified object is in contact with any of the custom objects

```
is_touching(object gameObject)
```

returns a boolean if there is contact

custom.is_point_in()

determines if a specified point is inside any of the custom objects

```
is_point_in(int x, int y)
```

returns nothing

custom.move()

moves every custom object by a factor of *x*, and *y*

```
move(int x, int y)
```

returns nothing

custom.delete_object()

deletes the specified object from the system

```
get_rect(object customObject)
```

returns nothing

© Copyright CPSuperstore 2018

Single Custom Object Changes

The following objects affect only a single custom object

Single Custom Object Properties

customObject.x

The x position of the custom object relative to the screen
int

customObject.y

The y position of the custom object relative to the screen
int

customObject.image

The path to the image used by the custom object
string

customObject.w

The width of the image of the custom object
int

customObject.h

The height of the image of the custom object
int

customObject.angle

The angle at which the object will move (if AI is True)
int

customObject.ai

If the object will follow the default AI algorithm to move
boolean

customObject.speed

The speed at which the object will travel at (if AI is True)
int

customObject.name

The name of the custom object
string

© Copyright CPSuperstore 2018

customObject.time_created

The UNIX time when the custom object was created

float

Single Custom Object Methods

customObject.get_rect()

returns the pygame rectangle (rect) object of the custom object

```
get_rect()
```

returns pygame rect object

customObject.is_touching()

detects if the specified object is touching the custom object

```
is_touching(object gameObject)
```

returns a boolean if the point is in the object

customObject.is_point_in()

detects if the specified point is in the custom object

```
is_point_in(tuple point)
```

returns a boolean if the point is in the object

© Copyright CPSuperstore 2018

World Border Changes

The world border is the white rectangle that surrounds the playable game area. No object can naturally leave this zone.

World Border Properties

world_border.x
: the x position of the top-left corner of the world border
int

world_border.y
: the y position of the top-left corner of the world border
int

world_border.w
: The width of the border
int

world_border.h
: The height of the world border
int

world_border.x2
: the x position of the bottom-right corner of the world border
int

world_border.y2
: the y position of the bottom-right corner of the world border
int

© Copyright CPSuperstore 2018

World Border Methods

world_border.move()

moves the entire world border by a factor of *x*, and *y*

```
move(int x, int y)
```

returns a boolean if the point is in the object

world_border.get_rect()

returns the pygame rect object of the world border

```
get_rect()
```

returns pygame rect object

world_border.get_range()

returns the opposite corners of the world border in the format `[(x, y), (x2, y2)]`

```
get_range()
```

returns list of points

world_border.is_inside()

detects if the specified object/points are inside the world border

```
is_inside(object gameObject=None, int x=None, int y=None)
```

returns if the specified paramtetes are inside the border as a boolean

© Copyright CPSuperstore 2018

Audio

Please Note. The best audio file format is *.ogg*, as it works best across all platforms, however, *.mp3* files work on Windows only.

Sound Objects

shoot

The sound object that is played when the player shoots
pygame.mixer.sound

Channel Map

The following table is a list of each Pygame.Mixer.Channel used in the game. There can be up to eight channels active in the game (channel 8 will have the number 7).

Channel Number	Usage
0	Story Audio
1	Background Music
2	Shoot Sound
3	Soup Collect Sound

Audio Players

audio.Audio()

This is a class which plays a sound file. When the sound is done, it will be replayed

```
Audio(int channel_number, string path, bool play_audio=True)
```

returns instance of the Audio class

story.play()

This function plays an audio file. If *update_screen* is True, the screen will turn black while the song plays. If not, the screen will keep what was last displayed.

```
play(
      display screen, string filename, bool update_screen=True,
      string path="rescources/audio/voice"
)
```

returns nothing

story.raw_play()

plays the specified audio file

```
raw_play(string filename)
```

returns nothing

© Copyright CPSuperstore 2018

Store Control

The store is the spot in the game where the user can spend hard-earned (or cheated in) coins to purchase upgrades.

The store has one editable option, which is the sale stack. This is a huge two-dimensional array of each item that is on sale, and its different sale properties.

Each item of the store list is another list (2D array). The list inside the list contains the details of each purchasable item.

The variable that stores this is called *items*. To access the cost of the first item in the store, the following command would need to be issued:

```
store.items[0][1]
```

Here is a table, which breaks down each item in the store.

Index	Description	Type
0	Display Name of the item	String
1	Cost of the item	Int
2	Internal name (do not touch this)	String
3	Maximum amount of purchases (-1 for infinite, -2 for blocked)	Int
4	Current number of purchases	Int

© Copyright CPSuperstore 2018

Other Methods

These methods are the general miscellaneous functions which make the game run.

dialog.alert()

displays an alert box with a specified message, pausing all other processes.
NOTE: *push_update* MUST be False! If not, there are some strange results.

```
alert(
      screen, string message, list key_listener, boolean push_update,
      int box_shrink
      )
```

returns key_listener key pushed

fade.fade_text()

Will display a black screen with red text that fades in and out. If "*skip*" is returned, the user has skipped the animation. Otherwise the animation has completed

```
fade_text(screen, string textToFadeOut, int delay)
```

returns user watch status as a string

screenshot()

takes a screenshot of the player's screen. Use the *show_message* argument to set if the screenshot message will be sent to the console

```
screenshot(bool show_message=True)
```

returns nothing

nuke()

Sends a nuke from the player. (Will kill all enemies in the level)

```
nuke()
```

returns nothing

optimizer()

Kills any threads that are dead. This is automatically run, but was listed for completeness

```
optimizer()
```

returns nothing

reset_world()

resets the world by clearing all bullets, returning the player to the original position, and resetting the player's angle

```
reset_world()
```

returns nothing

world_to_center()

moves the player back to the center of the map.

```
world_to_center()
```

returns nothing

© Copyright CPSuperstore 2018

game_over()

plays the game over sequence (shows points, credits, “You Loose”, and finally the menu)

```
game_over()
```

returns nothing

exit_confirm()

asks the player if they are sure they would like to return to the menu. This function DOES NOT return the player to the menu

```
exit_confirm()
```

returns a boolean if the player would like to exit

enable_commands()

enables the use of single player commands. No output message is created.

```
enable_commands()
```

returns nothing

disable_commands()

disables the use of single player commands. No output message is created. This does not re-enable score or story

```
disable_commands()
```

returns nothing

cmd.run_command()

Runs the specified command. This will only work if commands are enabled. See Appendix B for a list of commands.

```
run_command(string command)
```

returns command output as a string

config.get_controls()

Returns the dictionary of the user’s current key map

```
config.get_controls()
```

returns dict of keymap

credits.run()

runs the credit sequence. The *extra_lines* list is where you can put your own lines to be shown after all of the other credits are finished

```
run(
      Surface screen, Bool center=True, bool cary_over_audio=False,
      list extra_lines=None
)
```

returns nothing

© Copyright CPSuperstore 2018

hud.get_size()

Returns the width and height of the bottom green HUD.

```
get_size()
```

returns tuple

loading.load_screen()

Shows the loading screen in the middle of the screen, while the specified function runs. Please Note: The function cannot have any parameters. Pass in the function name WITHOUT the parentheses after the name. It is highly recommended to pass in the screen, although it is not required

```
load_screen(
      string message, function wait_function, Surface screen=None
)
```

returns nothing

scripts.get_name()

returns a random name from the enemy name pool.

```
get_name()
```

returns string name

scripts.lite_restriction ()

Displays the Lite Version Restriction.

```
lite_restriction(surface screen)
```

returns tuple

scripts.get_screenshot_name()

Returns the next available number to name a screenshot.

```
get_screenshot_name()
```

returns int

text_format.wrapline()

Returns a list of each line of the word wrapped message.

```
lite_restriction(string text, Pygame.Font font, int max_width)
```

returns list

© Copyright CPSuperstore 2018

Contact

If you have any inquiries about modding, or would like to request a change, or addition to the modding language, please do not hesitate to contact me at cpsuperstoreinc@gmail.com. You can also contact me at this email address to report bugs, and other features that do not work as they should. If you need an additional library to create your mod, again, please let me know, and I will look in to getting the library compatible with the mod.

Dump Files

In the event your mod crashes, a message will be displayed to the console that it has crashed, and a dump file will be generated. The file naming convention will be the name of the mod, and extension, followed by the word dump, and the Unix time of the crash. Here is an example.

```
mod_name.gwd_dump_1516939071.dump
```

From the file name, we know the mod file was *mod_name.gwd*. We also know that the crash happened at 1516939071. This translates to Friday, January 26, 2018 at 10:57:51 PM, GMT - 05:00.

The dump file will contain the Python stack traceback of the issue. Each mod lives in its own thread, and so a mod crashing will not have any affect the game. If we open the file above n any text editor (I will again recommend Notepad ++) ere is an example of a dump file.

```
1. Traceback (most recent call last):
2.   File "<string>", line 1, in <module>
3. ZeroDivisionError: integer division or modulo by zero
```

Starting from the second line and up will give us the details of the error. The second line indicates which line caused the error. In this case the stack traceback says

```
2. File "<string>", line 1, in <module>
```

This indicates that the error occurred in line 1 of the file (Indicated in red). The third line stores the error message. In this case, the error message is `ZeroDivisionError`. This means that the mod tried to divide by zero. The details can be found next to the message. In this case, it is `integer division or modulo by zero`. This confirms that the mod tried to divide by zero. If we look at the actual mod file, we will easily be able to find the issue.

```
1. print 1/0
```

Appendix A – Notepad ++

Throughout this guide, I recommend Notepad ++ multiple times. Notepad ++ is a free text editor, specially designed for writing code. It can be found at https://notepad-plus-plus.org. It is covered by the GPL license. It supports custom plugin development, custom translations, and the ability to share your plugins, and use plugins created by other people. Notepad ++ also has a great community around it.

At the time of this writing, there is no Notepad++ plugin for this modding language.

© Copyright CPSuperstore 2018

Appendix B – Command List

This is a table of all native commands which can be run in the console. Please Note: Commands MUST start with the tilde (~) character. If not, the command will be recognized as chat text. Please Note: Commands are NOT case sensitive. This means if you are trying to fire the nuke with console commands, `~nuke` will work the same as `~NukE`.

Name	Parameters	Description
`~allowcommands`	None	Enables console commands. This cannot be undone through the console
`~help`	None	Displays a list of commands
`~changecoins`	int amount	Changes the number of coins the player has by *amount*. To remove coins, use a negative number
`~bossfight`	None	Starts the game in boss mode. All mods will be reloaded
`~togglespectate`	None	Toggles if the player is in spectator mode
`~spawnmysterybox`	None	Spawns a mystery box
`~spawnsoup`	None	Spawns a soup can
`~nuke`	None	Fires A Nuke
`~clearcoins`	None	Clears All Of The Coins From The Level

© Copyright CPSuperstore 2018