DungeonBreath Manual

Making a Game

Nick Valentine

Contents

Part One

1. General

1.1 Input

In order to interact with the game in any way, shape or form, you will need to pass input into it. These inputs are defined in

Theorem 1.1.1 — Config file. GameDir/GameData/config.txt

This file is a simple key value store where the first word on any line is the key, and the second is the value. Input definitions follow a very simple domain specific language in order to be defined.

Theorem 1.1.2 — Input DSN. device:[input number]:[axis or button]:[min value]:button

device may be either k for keyboard, or j for joystick. If the device is keyboard, then don't pass any of the optional values. The configuration would look something like k:d. However if the input device is a joystick, all options are required as the input number is which joystick to look at (1-8)[hopefully this will be refactored out soon] axis or button tells how to treat the input, and min value is useful in the case of axis to present a dead zone which will prevent false positive triggering.

2. In Engine Creation

2.1 Tile Sets

2.1.1 Image layout

The engine for DungeonBreath allows for variable size sprites, and variable size tilesets. However it will tend to work best when all images are of a size that is a power of 2, and no smaller than 16x16. Example sizes can be 32x32, 64x64, 256x256, and they don't have to be square, they can also be 32x128 if necessary. In order to import your sprites into the engine, lay them out into one single sprite sheet per use case. A hero should have their own sheet, separate from other NPCs. The most restrictive use case of separating sprites by use case is levels, each level may only have one tileset in use at a time. Once the tilesheet is layed out properly, add it to:

Theorem 2.1.1 — Image data directory. GameDir/GameData/img/

2.1.2 The tileset tool

Then you may launch the game in dev mode and press the "Tile Sets" button. Choose New and provide an easy symbolic name, the name of the spritesheet with none of the path before the img directory (images may be placed inside of sub folders). And provide as the base size, the smallest unit of measure necessary to accurately select the smallest sprite.

Once finished press next and you will be in the tileset edit screen, accept will commit the accumulated tiles up to this point as a tileset ending with the tile selected by hitting accept. If there are no previously accumulated tiles, this will become a static tile. Static tiles do not have animations and will get some engine optimizations. In order to accumulate tiles for animations, use fire while hovering over the tile you wish to select. Use this method to select all except for the very last tile, use enter on the last tile to commit as a single tileset.

2.2 Level Creation

Launch the game in dev mode and press the "Level Editor" button. You may choose new, or from a list of existing levels. Once in the level editor:

1. accept + fire will remove a tile
2. accept + alt_fire will enter actor select mode
3. alt_fire will enter tile select mode
4. fire will place the currently selected item at the cursor's position
5. escape will enter the escape menu where the user may set the layer, collision type, save, exit, or return to editing
6. up will move the cursor up one block
7. down will move the cursor down one block
8. left will move the cursor left one block
9. right will move the cursor right one block
10. next will place a collision cube at the cursor

At this time in order to choose what events are triggered by collisions, you must manually edit the world file and add to that section. The dsn is:

Theorem 2.2.1 — action dsn. <type> <action> <target>

Where type is the integer associated with the collision type, action is either teleport or call, and target is either the new world to load, or the lua function to call.

3. Scripting

3.1 API

3.1.1 Audio

Functions:

$$music.play(< string > name) \rightarrow < void > \tag{3.1}$$

play a song located in "./GameData/sound/" with the filename name

$$music.stop() \rightarrow < void > \tag{3.2}$$

stop all currently playing music

$$music.playing() \rightarrow < string > \tag{3.3}$$

get the name of the currently playing song

$$music.set\_volume(< float > volume) \rightarrow < void > \tag{3.4}$$

set the volume

3.1.2 Core

config

$$config.get\_int(< string > name, < int > default) \rightarrow < int > \tag{3.5}$$

get an int configuration option, provide a default. if config.save() is called, and the default was used, the default will be persisted.

$$config.get\_sring(< string > name, < string > default) \rightarrow < string > \tag{3.6}$$

get an string configuration option, provide a default. if config.save() is called, and the default was used, the default will be persisted.

$$config.set(< string > name, < string|int > value) \rightarrow < void > \tag{3.7}$$

change a configuration option. if config.save() is called, the change will be persisted.

$$config.save() \rightarrow < void > \tag{3.8}$$

persist all configuration changes.

input

1. input.up
2. input.down
3. input.left
4. input.right
5. input.escape
6. input.accept
7. input.fire
8. input.alt_fire
9. input.num_keys

Button enumerations.

$$input.is\_key\_pressed() \rightarrow < num > \tag{3.9}$$

check if a button is pressed.

lang

$$lang.next() \rightarrow < string > \tag{3.10}$$

change the current language to the next one in the index. returns the language name.

$$lang.prev() \rightarrow < string > \tag{3.11}$$

change the current language to the previous one in the index. returns the language name.

logger

$$logger.debug(< string > value, ...) \rightarrow < void > \tag{3.12}$$

log at the debug level.

$$logger.info(< string > value, ...) \rightarrow < void > \tag{3.13}$$

log at the info level.

$$logger.warn(< string > value, ...) \rightarrow < void > \tag{3.14}$$

log at the warn level.

$$logger.error(< string > value, ...) \rightarrow < void > \tag{3.15}$$

log at the error level.

index

$$index.get(< string > path) \rightarrow < Index > \tag{3.16}$$

get an Index object that references path, all indexes got must be released.

$$index.release(< Index > idx) \rightarrow < void > \tag{3.17}$$

release a Index object so it can be removed from memory.

$$index.add(< Index > idx, < string > value) \rightarrow < void > \tag{3.18}$$

add a value to this index, changes must be saved in order to be reflected onto disk.

$$index.remove(< Index > idx, < string > value) \rightarrow < void > \tag{3.19}$$

remove a value from this index, changes must be saved in order to be reflected onto disk.

$$index.save(< Index > idx) \rightarrow < void > \tag{3.20}$$

save all changes made to this index.

$$index.all(< Index > idx) \rightarrow < []string > \tag{3.21}$$

get all items in this index.

imgui

This library is a dev tool for ui's that normal users will not see.

$$imgui.start(< string > name) \rightarrow < void > \tag{3.22}$$

create a new imgui window.

$$imgui.stop() \rightarrow < void > \tag{3.23}$$

end the current imgui window.

$$imgui.start\_child(< string > name, < vec2 > pos) \rightarrow < void > \tag{3.24}$$

create a new imgui sub window.

$$imgui.stop\_child() \rightarrow < void > \tag{3.25}$$

end a imgui sub window.

$$imgui.button(< string > name) \rightarrow < bool > \tag{3.26}$$

create a new imgui button.

$$imgui.checkbox(<string> name, <bool> checked) \rightarrow <bool> \tag{3.27}$$

create a new imgui checkbox.

$$imgui.progressbar(<float> value) \rightarrow <void> \tag{3.28}$$

create a new imgui progressbar.

$$imgui.text(<string> value) \rightarrow <void> \tag{3.29}$$

create a new imgui text.

$$imgui.input\_text(<string> label, <string> value) \rightarrow <string> \tag{3.30}$$

create a new imgui text input.

$$imgui.input\_int(<string> label, <int> value) \rightarrow <int> \tag{3.31}$$

create a new imgui int input.

$$imgui.input\_float(<string> label, <float> value) \rightarrow <float> \tag{3.32}$$

create a new imgui float input.

$$imgui.image\_button(<Sprite> image) \rightarrow <bool> \tag{3.33}$$

create a new imgui image button.

$$imgui.listbox(<string> label, <int> current, <[]string> values) \rightarrow <int> \tag{3.34}$$

create a new imgui listbox, returns currently selected.

3.1.3 Play

world

$$world.change\_level(<World> w, <string> name, <vec2> player\_location) \rightarrow <void> \tag{3.35}$$

request a new level load.

$$world.get\_actorman(<World> w) \rightarrow <ActorManager> \tag{3.36}$$

get the world's actormanager.

$$world.draw(< World > w, < RenderWindow > win) \rightarrow < void > \tag{3.37}$$

draw the world.

$$world.draw\_layer(< World > w, < RenderWindow > win, < int > layer) \rightarrow < void > \tag{3.38}$$

draw the a layer of world.

$$world.draw\_actors(< World > w, < RenderWindow > win) \rightarrow < void > \tag{3.39}$$

draw the actors in this world.

$$world.set\_edit\_mode(< World > w, < bool > e) \rightarrow < void > \tag{3.40}$$

change the mode the world is in to be more usable for editing the world.

$$world.save\_edits(< World > w) \rightarrow < void > \tag{3.41}$$

save all changes made to this world back onto the original world file.

$$world.add\_actor(< World > w, < string > name, < vec2 > pos) \rightarrow < void > \tag{3.42}$$

add an actor into this world.

$$world.add\_collision(< World > w, < int > layer, < vec2 > pos) \rightarrow < void > \tag{3.43}$$

add a collision into this world.

$$world.set\_tile(< World > w, < Tile > tile, < int > layer, < vec2 > pos) \rightarrow < void > \tag{3.44}$$

change a tile in this world.

$$world.remove\_tile(< World > w, < int > layer, < vec2 > pos) \rightarrow < void > \tag{3.45}$$

remove a tile from this world.

$$world.get\_size(< World > w) \rightarrow < vec2 > \tag{3.46}$$

get the size in tiles of this world.

$$world.get\_tileset(< World > w) \rightarrow < string > \tag{3.47}$$

get the name of the tileset of this world.

$$world.get\_script\_name(< World > w) \rightarrow < string > \tag{3.48}$$

get the name of the script of this world.

Actor

$$actor.get\_rect(< Actor > a) \rightarrow < rect > \tag{3.49}$$

get the actor's collision rect

$$actor.set\_scale(< Actor > a, < vec2 > v) \rightarrow < void > \tag{3.50}$$

set the actor's scale

$$actor.set\_origin(< Actor > a, < vec2 > v) \rightarrow < void > \tag{3.51}$$

set the actor's origin

$$actor.set\_collision\_bounds(< Actor > a, < vec2 > size) \rightarrow < void > \tag{3.52}$$

set the actor's collision bounds

$$actor.get\_velocity(< Actor > a) \rightarrow < vec2 > \tag{3.53}$$

get the actor's velocity

$$actor.set\_velocity(< Actor > a, < vec2 > v) \rightarrow < void > \tag{3.54}$$

set the actor's velocity

$$actor.set\_tileset(< Actor > a, < int > t) \rightarrow < void > \tag{3.55}$$

set the actor's tileset

$$actor.pause\_anim(< Actor > a) \rightarrow < void > \tag{3.56}$$

pause the actor's animation

$$actor.play\_anim(< Actor > a) \rightarrow < void > \tag{3.57}$$

play the actor's animation

$$actor.reset\_anim(< Actor > a) \rightarrow < void > \tag{3.58}$$

reset the actor's animation

$$actor.draw(< Actor > a, < Window > w) \rightarrow < void > \tag{3.59}$$

draw the actor

Actor Manager

$$actor\_manager.spawn(<ActorManager> a, <string> name, <vec2> pos) \rightarrow <actor\_handle> \tag{3.60}$$

spawn an actor

$$actor\_manager.remove(<ActorManager> a, <int> actor\_handle) \rightarrow <void> \tag{3.61}$$

remove an actor

$$actor\_manager.clear(<ActorManager> a) \rightarrow <void> \tag{3.62}$$

clear out all actors

$$actor\_manager.set\_camera\_target(<ActorManager> a, <actor\_handle> h) \rightarrow <void> \tag{3.63}$$

set the actor that the camera should follow

$$actor\_manager.get\_camera\_target(<ActorManager> a) \rightarrow <Actor> \tag{3.64}$$

get the actor that the camera should follow

$$actor\_manager.get\_player(<ActorManager> a) \rightarrow <Actor> \tag{3.65}$$

get the actor that represents the player

$$actor\_manager.set\_player(<ActorManager> a, <actor\_handle> h) \rightarrow <void> \tag{3.66}$$

set the actor that represents the player

3.1.4 Render

Sprite Manager

$$sprite\_manager.get(<string> texture\_name, <int> sprite\_name) \rightarrow <SpriteManager> \tag{3.67}$$

get a new sprite manager, release should be called on all object yeilded from this method

$$sprite\_manager.release(<SpriteManager> s) \rightarrow <SpriteManager> \tag{3.68}$$

frees this object from memory

$$sprite\_manager.make\_sprite(<SpriteManager> s, <vec2> pos, <vec2> size) \rightarrow <Sprite> \tag{3.69}$$

creates a sprite on this sprite manager

$$sprite\_manager.remove\_sprite(<SpriteManager> s, <Sprite> spr) \rightarrow <void> \tag{3.70}$$

removes a sprite from this sprite manager

Sprite

$$sprite.set\_position(< Sprite > s, < vec2 > pos) \rightarrow < void > \tag{3.71}$$

sets the position of a sprite

$$sprite.set\_scale(< Sprite > s, < vec2 > pos) \rightarrow < void > \tag{3.72}$$

sets the scale of a sprite

$$sprite.draw(< Sprite > s, < Window > win) \rightarrow < void > \tag{3.73}$$

draws a sprite to the window

TileSet

$$tile\_set.get(< string > tileset\_name) \rightarrow < TileSet > \tag{3.74}$$

get a new tileset, release should be called on all object yeilded from this method

$$tile\_set.release(< TileSet > t) \rightarrow < void > \tag{3.75}$$

frees this object from memory

$$tile\_set.keys(< TileSet > t) \rightarrow < []int > \tag{3.76}$$

get all of the possible keys for tiles that can be spawned

$$tile\_set.get\_tile(< TileSet > t, < int > key) \rightarrow < Tile > \tag{3.77}$$

get a tile from this tileset, release should be called on all object yeilded from this method

Tile

$$tile.release(< Tile > t) \rightarrow < void > \tag{3.78}$$

frees this object from memory

$$tile.update(< Tile > t, < int > delta) \rightarrow < void > \tag{3.79}$$

update this tile

$$tile.draw(< Tile > t, < Window > window) \rightarrow < void > \tag{3.80}$$

draw this tile to the window

$$tile.play(< Tile > t) \rightarrow < void > \tag{3.81}$$

play this tile’s animation.

$$tile.pause(< Tile > t) \rightarrow < void > \tag{3.82}$$

pause this tile's animation.

$$tile.reset(< Tile > t) \rightarrow < void > \tag{3.83}$$

reset this tile's animation.

$$tile.set\_location(< Tile > t, < vec2 > pos) \rightarrow < void > \tag{3.84}$$

set this tile's location

$$tile.get\_location(< Tile > t) \rightarrow < vec2 > \tag{3.85}$$

get this tile's location

$$tile.set\_scale(< Tile > t, < vec2 > s) \rightarrow < void > \tag{3.86}$$

set this tile's scale

$$tile.set\_origin(< Tile > t, < vec2 > o) \rightarrow < void > \tag{3.87}$$

set this tile's origin

$$tile.get\_icon(< Tile > t) \rightarrow < Sprite > \tag{3.88}$$

set this tile's icon

3.1.5 Scene

$$scene.push(< Scene > s, < string > name) \rightarrow < void > \tag{3.89}$$

push a new scene onto the scene stack

$$scene.pop(< Scene > s) \rightarrow < void > \tag{3.90}$$

pop this scene from the scene stack

$$scene.get\_menu(< Scene > s) \rightarrow < Menu > \tag{3.91}$$

get the menu owned by this scene

$$scene.get\_size(< Scene > s) \rightarrow < vec2 > \tag{3.92}$$

get the size of this scene

$$scene.reset\_camera(< Scene > s) \rightarrow < void > \tag{3.93}$$

reset the scene's camera back to 0

$$scene.get\_camera\_center(<Scene> s) \rightarrow <vec2> \tag{3.94}$$

get the camera's center

$$scene.move\_camera(<Scene> s, <vec2> diff) \rightarrow <void> \tag{3.95}$$

move the camera by diff

$$scene.set\_viewport(<Scene> s, <rect> r) \rightarrow <void> \tag{3.96}$$

set the view to r

$$scene.get\_viewport(<Scene> s) \rightarrow <rect> \tag{3.97}$$

get the scene's viewport

$$scene.zoom\_camera(<Scene> s, <float> mult) \rightarrow <void> \tag{3.98}$$

zoom the scene by mult

$$scene.apply\_view(<Scene> s, <Window> window) \rightarrow <void> \tag{3.99}$$

apply the scene's camera

$$scene.init\_world(<Scene> s) \rightarrow <void> \tag{3.100}$$

initialize the scene's world

$$scene.get\_world(<Scene> s) \rightarrow <World> \tag{3.101}$$

get the scene's world

$$scene.draw(<Scene> s, <Window> win) \rightarrow <void> \tag{3.102}$$

draw the scene

3.1.6 Shape

Rectangle Shape

$$rectangle\_shape.get() \rightarrow <RectangleShape> \tag{3.103}$$

get a rectangle, release should be called on all object yeilded from this method

$$rectangle\_shape.release(<RectangleShape> r) \rightarrow <void> \tag{3.104}$$

free this object from memory

$$rectangle\_shape.set\_size(<RectangleShape> r, <vec2> s) \rightarrow <void> \tag{3.105}$$

set this objects size

$$rectangle\_shape.set\_position(<RectangleShape> r, <vec2> p) \rightarrow <void> \tag{3.106}$$

set this objects position

$$rectangle\_shape.set\_fill\_color(<RectangleShape> r, <rgb[a]> p) \rightarrow <void> \tag{3.107}$$

set this objects fill color

$$rectangle\_shape.set\_outline\_color(<RectangleShape> r, <rgb[a]> p) \rightarrow <void> \tag{3.108}$$

set this objects outline color

$$rectangle\_shape.set\_outline\_thickness(<RectangleShape> r, <float> s) \rightarrow <void> \tag{3.109}$$

set this objects outline thickness

$$rectangle\_shape.draw(<RectangleShape> r, <Window> win) \rightarrow <void> \tag{3.110}$$

draw this object to window

3.1.7 UI

Menu

$$menu.set\_current(<Menu> m, <MenuItem> mi) \rightarrow <void> \tag{3.111}$$

set the currently hovered menu item

$$menu.add\_sprite\_button(<Menu> m, <string> tag, <rect> pos, <string> tex, <rect> normal, <rect> hover, \tag{3.112}$$

insert a spritebutton into this menu

$$menu.add\_label(<Menu> m, <rect> pos, <string> content\_key) \rightarrow <MenuItem> \tag{3.113}$$

insert a label into this menu, content_key is a lookup into the active lang file.

$$menu.add\_label\_raw(<Menu> m, <rect> pos, <string> content) \rightarrow <MenuItem> \tag{3.114}$$

insert a label into this menu

$$menu.has\_signal(<Menu> m) \rightarrow <bool> \tag{3.115}$$

see if this menu has a signal to be processed

$$menu.signal\_tag(< Menu > m) \rightarrow < string > \tag{3.116}$$

get the signals ui tag

$$menu.signal\_str(< Menu > m) \rightarrow < string > \tag{3.117}$$

get the signals string message

$$menu.signal\_int(< Menu > m) \rightarrow < int > \tag{3.118}$$

get the signals int message

$$menu.clear(< Menu > m) \rightarrow < void > \tag{3.119}$$

empty out this menu

MenuItem

$$menu\_item.raw(< MenuItem > m) \rightarrow < MenuItem > \tag{3.120}$$

get the underlying object in this menu item

$$menu\_item.set\_up(< MenuItem > m, < MenuItem > o) \rightarrow < void > \tag{3.121}$$

set the relationship between two buttons

$$menu\_item.set\_down(< MenuItem > m, < MenuItem > o) \rightarrow < void > \tag{3.122}$$

set the relationship between two buttons

$$menu\_item.set\_left(< MenuItem > m, < MenuItem > o) \rightarrow < void > \tag{3.123}$$

set the relationship between two buttons

$$menu\_item.set\_right(< MenuItem > m, < MenuItem > o) \rightarrow < void > \tag{3.124}$$

set the relationship between two buttons

$$menu\_item.pair\_items(< MenuItem > m, < MenuItem > o, < int > direction) \rightarrow < void > \tag{3.125}$$

set the relationship between two buttons

$$menu\_item.get\_tag(< MenuItem > m) \rightarrow < string > \tag{3.126}$$

get the tag set when creating this item

Label

$$label.get(< rect > p, < string > contents) \rightarrow < Label > \tag{3.127}$$

create a new unmanaged label, all labels created by this method should be released

$$label.release(< Label > l) \rightarrow < void > \tag{3.128}$$

release an unmanaged label

$$label.set\_position(< Label > l, < rect > pos) \rightarrow < void > \tag{3.129}$$

set this objects position

$$label.set\_string(< Label > l, < string > contents) \rightarrow < void > \tag{3.130}$$

set this objects string

$$label.set\_size(< Label > l, < float > size) \rightarrow < void > \tag{3.131}$$

set this objects char size

$$label.draw(< Label > l, < Window > win) \rightarrow < void > \tag{3.132}$$

draw this object

3.2 Hooks

There are three main script hooks into this engine.

3.2.1 Actors

An actor is anything that moves while on screen, actors are composed of a global table named 'me' which is to contain a tileset string variable denoting which tileset the actor wishes to use. An update function of the form:

$$me.update(< int > delta) \rightarrow < void > \tag{3.133}$$

an init function:

$$me.init() \rightarrow < void > \tag{3.134}$$

a draw function, which is optional in order to override default drawing behavior:

$$me.draw(< Window > win) \rightarrow < void > \tag{3.135}$$

and a release function to release resources when the actor is destructed

$$me.release() \rightarrow < void > \tag{3.136}$$

3.2.2 Levels

Levels are scriptable add potential events to special collisions, in a level you can define one script to run, then in the collision section, map collision types to functions inside a me table inside that script. the functions should take no parameters and have no return. Like actor: draw is overridable.

3.2.3 Scenes

Scenes are your game state. Everything is a scene, menus, submenus, even the game itsself. scenes should export a me table as well, the me table should have the functions:

$$me.update(< int > delta) \rightarrow < void > \tag{3.137}$$

an init function:

$$me.init() \rightarrow < void > \tag{3.138}$$

a draw function, which is optional in order to override default drawing behavior:

$$me.draw(< Window > win) \rightarrow < void > \tag{3.139}$$

a release function to release resources when the actor is destructed

$$me.release() \rightarrow < void > \tag{3.140}$$

a wakeup function for when this scene comes back into focus

$$me.wakeup() \rightarrow < void > \tag{3.141}$$

and a sleep function for when this scene goes out of focus

$$me.sleep() \rightarrow < void > \tag{3.142}$$