# TP FINAL PROGRAMACION 1

Generated by Doxygen 1.9.2

# Chapter 1

# TPFINALGRUPO1

En el presente repositorio se encuentra el trabajo practico final realizado por el grupo 1 del cuatrimestre 2 de 2020, para ser presentado en febrero

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 bloque_t Struct Reference

### Public Attributes

- fisica_t **fisica**
- int **identificador**
- int **sprite**

The documentation for this struct was generated from the following file:

- entidades.h

## 4.2 bufferRecursos_t Struct Reference

### Public Attributes

- int **imageQuant**
- int **soundQuant**
- int **fontQuant**
- image_t * **image**
- sonido_t * **sound**
- fuente_t * **font**

The documentation for this struct was generated from the following file:

- matiasBrosGame.h

## 4.3 dcoord_t Struct Reference

### Public Attributes

- uint8_t **x**
- uint8_t **y**

The documentation for this struct was generated from the following file:

- disdrv.h

## 4.4 enemigo_t Struct Reference

### Public Attributes

- fisica_t **fisica**
- pthread_t **enemyIA**
- int **estado**
- int **identificador**
- int **sprite**
- int **moveAgain**
- void ∗(∗ **funcionMovimiento** )(void ∗)

The documentation for this struct was generated from the following file:

- entidades.h

## 4.5 entidades_t Struct Reference

### Public Attributes

- jugador_t **jugador**
- enemigo_t ∗ **enemigos**
- bloque_t ∗ **bloques**

The documentation for this struct was generated from the following file:

- entidades.h

## 4.6 estadoJuego_t Struct Reference

### Public Attributes

- int **maxTopScoreEntries**
- int **bestScores** [MAXTOPSCORES]
- char **bestScoresName** [MAXTOPSCORES][MAXBESTSCORENAME]
- estadosjuego_t **state**
- int **menuSelection**
- int **pauseSelection**
- entidades_t **entidades**
- entidades_t **defaultEntities**
- level_t **level**
- bufferRecursos_t **buffer**
- gameUI_t **gameUI**
- char * **pPlayerName**

The documentation for this struct was generated from the following file:

- matiasBrosGame.h

## 4.7 fisica_t Struct Reference

### Public Attributes

- float **posx**
- float **posy**
- int **ancho**
- int **alto**
- float **velx**
- float **vely**

The documentation for this struct was generated from the following file:

- entidades.h

## 4.8 gameUI_t Struct Reference

### Public Attributes

- int **score**
- int **coins**
- int **level**
- int **time**

The documentation for this struct was generated from the following file:

- matiasBrosGame.h

## 4.9 image_menu_t Struct Reference

### Public Attributes

- float **x**
- float **y**
- float **scale**

The documentation for this struct was generated from the following file:

- menu.c

## 4.10 jcoord_t Struct Reference

### Public Attributes

- int8_t **x**
- int8_t **y**

The documentation for this struct was generated from the following file:

- joydrv.h

## 4.11 jugador_t Struct Reference

### Public Attributes

- fisica_t **fisica**
- double **angleRotation**
- int **vidas**
- int **sobreBloque**
- int **isMoving**
- int **powerUpsState**
- int **estado**
- int **sprite**

The documentation for this struct was generated from the following file:

- entidades.h

## 4.12 level_t Struct Reference

### Public Attributes

- int ** **level**
- int **levelWidht**
- int **levelHeight**

The documentation for this struct was generated from the following file:

- matiasBrosGame.h

## 4.13 menu_t Struct Reference

### Public Attributes

- int **imgQuant**
- int **textQuant**
- image_menu_t * **imgMenu**
- text_menu_t * **textMenu**

The documentation for this struct was generated from the following file:

- menu.c

## 4.14 sound Struct Reference

```
#include <audio.h>
```

### Public Attributes

- uint32_t **length**
- uint32_t **lengthTrue**
- Uint8 ** **bufferTrue**
- uint8_t * **buffer**
- uint8_t **loop**
- uint8_t **fade**
- uint8_t **free**
- uint8_t **volume**
- SDL_AudioSpec **audio**
- struct sound * **next**

### 4.14.1 Detailed Description

Queue structure for all loaded sounds

The documentation for this struct was generated from the following file:

- audio.h

## 4.15 text_menu_t Struct Reference

### Public Attributes

- int **r**
- int **g**
- int **b**
- float **x**
- float **y**
- char **word** [MAXMENUWORDSIZE]

The documentation for this struct was generated from the following file:

- menu.c

## 4.16 timer Struct Reference

### Public Attributes

- int **ID**
- float **secondsPerTick**
- int **isRunning**
- int **isPaused**
- void(∗ **funtionToExecute** )(void ∗param)
- struct timer ∗ **next**

The documentation for this struct was generated from the following file:

- times.h

## 4.17 wave_t Struct Reference

### Public Attributes

- int **offsetX**
- int **moveDelay**
- int **offsetXRecord**
- float **scale**

The documentation for this struct was generated from the following file:

- level.c

# Chapter 5

# File Documentation

## 5.1 allegro.h File Reference

Headers necesarios utilizados por allegro.

## 5.2 allegroLib.h File Reference

Funciones de carga de datos e inicializacion relacionadas a Allegro. Tambien carga de highscores.

### Functions

- int cargarSonidosMenu (sonido_t **sonido)

  *Carga todos los sonidos conociendo la cantidad y su path desde soundsData.txt.*
- void destroyResources (bufferRecursos_t *resourcesBuffer)

  *Libera el espacio reservado para las imagenes, el sonido y las fuentes.*
- int inicializarAllegro ()

  *Inicializa tod0 lo necesario para luego utilizar allegro en el programa.*
- int cargarTexturasMenu (image_t **textura)

  *Carga todas las texturas conociendo la cantidad y su path desde texturesData.txt.*
- int cargarFuentesMenu (fuente_t **fuente)

  *Carga todas las fuentes conociendo la cantidad y su path desde fontsData.txt.*
- int loadGameState (estadoJuego_t *gameState)

  *Se carga la lista de high scores al programa.*

### 5.2.1 Function Documentation

#### 5.2.1.1 cargarFuentesMenu()

```
int cargarFuentesMenu (
            fuente_t ** fuente )
```

**Parameters**

| | |
|---|---|
| ***fuente* | puntero a la estructura fuente_t dentro de gameState |

**Returns**

Devuelve 0 si se cargo bien, sino -1

#### 5.2.1.2 cargarSonidosMenu()

```
int cargarSonidosMenu (
            sonido_t ** sonido )
```

**Parameters**

| | |
|---|---|
| ***sonido* | puntero a la estructura sonido_t dentro de gameState |

**Returns**

Devuelve 0 si se cargo bien, sino -1

#### 5.2.1.3 cargarTexturasMenu()

```
int cargarTexturasMenu (
            image_t ** textura )
```

**Parameters**

| | |
|---|---|
| ***textura* | puntero a la estructura image_t dentro de gameState |

**Returns**

Devuelve 0 si se cargo bien, sino -1

#### 5.2.1.4 destroyResources()

```
void destroyResources (
            bufferRecursos_t * resourcesBuffer )
```

**Parameters**

| | |
|---|---|
| *∗resourcesBuffer* | puntero a la estructura bufferRecursos_t la cual contiene todos los datos que destruira |

#### 5.2.1.5 inicializarAllegro()

```
int inicializarAllegro ( )
```

**Returns**

Devuelve 0 si se cargo bien, sino -1

Inicializa tod0 lo necesario para luego utilizar allegro en el programa.

file allegroLib.c

#### 5.2.1.6 loadGameState()

```
int loadGameState (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | Puntero a gameState |

**Returns**

Devuelve 0 si se cargo bien, sino -1

## 5.3 animacion.h File Reference

Control de animaciones y timers correspondientes.

### Functions

- void ∗ animar (void ∗gs)

  *Se encarga de controlar las animaciones de la mayor parte del programa.*
- void stopInGameAnimations ()

  *Se encarga de frenar todos los timers correspondientes a las animaciones.*
- void startInGameAnimations ()

  *Se encarga de inicializar todos los timers necesarios para las animaciones.*

### 5.3.1 Function Documentation

#### 5.3.1.1 animar()

```
void* animar (
            void * gs )
```

**Parameters**

| | |
|---|---|
| *gs* | Puntero que debe referenciar a la estructura principal del programa |

## 5.4 configuracion.h File Reference

Archivo principal de configuracion del juego.

## 5.5 data.h File Reference

Funciones para la lectura de archivos .txt localizados en la carpeta data.

### Enumerations

- enum **menuImages** {
  **FONDOMENU** , **CARTELMENU** , **FLECHAMENU** , **LEVELSELECTORIDLE** ,
  **LEVELSELECTORLEFT** , **LEVELSELECTORRIGHT** , **SCORETABLEIMG** }
- enum **menuTexts** { **SELECTTEXT** , **LEVELTEXT** , **SCORETEXT** , **EXITTEXT** }
- enum **levelImages** {
  **MATIASIDLESPRITE** = SCORETABLEIMG+1 , **PLAYERSWIMMING1** , **PLAYERSWIMMING2** , **PLAYERSWIMMING3** ,
  **PLAYERSWIMMING4** , **MATIASIDLEBIGSPRITE** , **PLAYERSWIMMINGBIG1** , **PLAYERSWIMMINGBIG2** ,
  **PLAYERSWIMMINGBIG3** , **PLAYERSWIMMINGBIG4** , **CHEEPCHEEPSPRITE1** , **CHEEPCHEEPSPRITE2** ,
  **CHEEPCHEEPSSLOWSPRITE1** , **CHEEPCHEEPSSLOWSPRITE2** , **BLOOPERSPRITE1** , **BLOOPERSPRITE2** ,
  **ALGASPRITE1** , **ALGASPRITE2** , **PISOSPRITE** , **WAVESPRITE** ,
  **BUBBLESPRITE** , **COINSPRITE1** , **COINSPRITE2** , **COMMONMUSHROOMSPRITE** ,
  **PIPEMIDDLESPRITE** , **PIPETOPSPRITE** , **CASTELSPRITE** }
- enum **fuentes** {
  **SUPERMARIOFONT60** , **SUPERMARIOFONT80** , **SUPERMARIOFONT100** , **SUPERMARIOFONT120** ,
  **SUPERMARIOFONT140** }
- enum **audio** {
  **SUPERMARIOTHEME** , **UNDERWATERTHEME** , **PICKUPCOIN** , **JUMPSMALL** ,
  **PAUSEGAME** , **ENTERPIPE** , **GAMEOVERSOUND** , **MARIODIES** ,
  **POWERUPSOUND** , **ONEUP** , **WARNINGTIMEOUT** }

## Functions

- int openGameStateFile (FILE **gameStateData)

  *Lectura del archivo estadojuegoData.txt donde se encuentran los highscores.*
- int openTexturesFile (FILE **texturaData)

  *Lectura del archivo texturesData.txt donde se encuentran las texturas.*
- int openFontsFile (FILE **fontsData)

  *Lectura del archivo fontsData.txt donde se encuentran las fuentes.*
- int openMenuData (FILE **imageMenuData, FILE **textMenuData)

  *Lectura del archivo imgMenuData.txt y textMenuData.txt donde se encuentran las imagenes y textos del menu.*
- int openLevelData (FILE **levelData, int id)

  *Lectura del archivo levelX.txt donde se encuentra un nivel en particular.*
- int openSoundsFile (FILE **soundData)

  *Lectura del archivo soundsData.txt donde se encuentran los sonidos.*
- int getMaxLevelsAvailable ()

  *Lee el nombre de los archivos .txt de la forma level_.txt y devuelve la cantidad de nivele disponibles.*

### 5.5.1 Function Documentation

#### 5.5.1.1 getMaxLevelsAvailable()

```
int getMaxLevelsAvailable ( )
```

**Returns**

Retorna la cantidad de niveles disponibles

#### 5.5.1.2 openFontsFile()

```
int openFontsFile (
            FILE ** fontsData )
```

**Parameters**

| | |
|---|---|
| ***fontsData* | puntero a FILE donde se localizara el archivo abierto |

**Returns**

Devuelve 0 si la operacion fue correcta, en caso contrario devuelve 1

#### 5.5.1.3 openGameStateFile()

```
int openGameStateFile (
            FILE ** gameStateData )
```

**Parameters**

| | |
|---|---|
| ***gameStateData* | puntero a FILE donde se localizara el archivo abierto |

**Returns**

Devuelve 0 si la operacion fue correcta, en caso contrario devuelve 1

#### 5.5.1.4 openLevelData()

```
int openLevelData (
            FILE ** levelData,
            int id )
```

**Parameters**

| | |
|---|---|
| ***levelData* | puntero a FILE donde se localizara el archivo abierto |
| *id* | El nivel a abrir, un numero entre 0 para el nivel 1, 1 para el nivel 2 y 2 para el nivel 3. |

**Returns**

Devuelve 0 si la operacion fue correcta, en caso contrario devuelve 1

#### 5.5.1.5 openMenuData()

```
int openMenuData (
            FILE ** imageMenuData,
            FILE ** textMenuData )
```

**Parameters**

| | |
|---|---|
| ***imageMenuData* | puntero a FILE donde se localizara el archivo abierto |
| ***textMenuData* | puntero a FILE donde se localizara el archivo abierto |

**Returns**

Devuelve 0 si la operacion fue correcta, en caso contrario devuelve 1

#### 5.5.1.6 openSoundsFile()

```
int openSoundsFile (
            FILE ** soundData )
```

**Parameters**

| | |
|---|---|
| ***soundData* | puntero a FILE donde se localizara el archivo abierto |

**Returns**

Devuelve 0 si la operacion fue correcta, en caso contrario devuelve 1

#### 5.5.1.7 openTexturesFile()

```
int openTexturesFile (
            FILE ** texturaData )
```

**Parameters**

| | |
|---|---|
| ***texturaData* | puntero a FILE donde se localizara el archivo abierto |

**Returns**

Devuelve 0 si la operacion fue correcta, en caso contrario devuelve 1

## 5.6 disdrv.h File Reference

Display Driver.

### Classes

- struct dcoord_t

### Enumerations

- enum **dlevel_t** { **D_OFF** , **D_ON** }

### Functions

- void disp_init (void)

  *Inicializa el display. Debe llamarse UNA SOLA VEZ al empezar a usar el display.*
- void disp_clear (void)

  *Borra enteramente tanto el display como el buffer (ver disp_write)*
- void disp_write (dcoord_t coord, dlevel_t val)

  *Escribe al buffer, NO al display.*
- void disp_update (void)

  *Actualiza ttodo el display con el contenido del buffer.*

### 5.6.1 Detailed Description

**Author**

Daniel Jacoby, Nicolas Magliola y Pablo Vacatello

### 5.6.2 Function Documentation

#### 5.6.2.1 disp_write()

```
void disp_write (
            dcoord_t coord,
            dlevel_t val )
```

**Parameters**

| | |
|---|---|
| *coord* | Estructura de tipo dcoord_t, que contiene las coordenadas del punto al que se quiere escribir. Recordar que (DISP_MIN $<$ x $<$ DISP_MAX_X) y que (DISP_MIN $<$ y $<$ DISP_MAX_Y). |
| *val* | valor que se escribirá para el punto indicado por coord. Puede ser D_OFF o D_ON. |

**Returns**

Descripcion valor que devuelve

## 5.7 entidades.h File Reference

definicion de entidades_t y funciones relacionadas a la misma.

### Classes

- struct fisica_t
- struct jugador_t
- struct enemigo_t
- struct bloque_t
- struct entidades_t

### Enumerations

- enum **CHARACTERSTATE** {
  **ALIVE** , **DEAD** , **SLEPT** , **ALMOSTDEAD** ,
  **INVULNERABLE** }

## Functions

- void startEnemy (enemigo_t *thisEnemy)

  *Inicializa el thread de un enemigo correspondiente a su movimiento y pone su estado en ALIVE.*
- void setClosestPlayer (jugador_t *player)

  *Se utiliza para determinar al jugador actual (matias), es necesaria para que luego los blooper puedan moverse adecuadamente.*
- void * blooper (void *enemy)

  *Funcion que determina, en un thread particular, el movimiento de un blooper.*
- void * cheepcheep (void *enemy)

  *Funcion que determina, en un thread particular, el movimiento de un cheepcheep.*

### 5.7.1 Function Documentation

#### 5.7.1.1 blooper()

```
void* blooper (
            void * enemy )
```

**Parameters**

| | |
|---|---|
| **enemy* | recibe un puntero al enemigo del cual modificaremos su movimiento |

#### 5.7.1.2 cheepcheep()

```
void* cheepcheep (
            void * enemy )
```

**Parameters**

| | |
|---|---|
| **enemy* | recibe un puntero al enemigo del cual modificaremos su movimiento |

#### 5.7.1.3 setClosestPlayer()

```
void setClosestPlayer (
            jugador_t * player )
```

**Parameters**

| | |
|---|---|
| **player* | Recibe un puntero a jugador_t player |

#### 5.7.1.4 startEnemy()

```
void startEnemy (
            enemigo_t * thisEnemy )
```

**Parameters**

| | |
|---|---|
| *∗thisEnemy* | puntero a estructura enemigo_t de donde obtendra la funcion de su movimiento y el id del thread |

## 5.8 fisica.h File Reference

Libreria para motor de fisica.

### Functions

- void ∗ fisica (void ∗entrada)

  *Thread principal donde se maneja la fisica del videojuego.*
- void movePlayer (int direction, void ∗player)

  *Thread principal donde se maneja la logica del videojuego.*

### 5.8.1 Function Documentation

#### 5.8.1.1 fisica()

```
void* fisica (
            void * entrada )
```

**Parameters**

| | |
|---|---|
| *entrada* | puntero al estado del juego |

#### 5.8.1.2 movePlayer()

```
void movePlayer (
            int direction,
            void * player )
```

**Parameters**

| | |
|---|---|
| *direction* | Evento de tecla de direccion presionada. Ej. DOWNIZQUIERDA, DOWNDERECHA, etc |
| *player* | puntero al jugador (jugador_t∗) |

## 5.9 gamelogic.h File Reference

Uno de los threads principales donde se maneja la logica del videojuego.

### Functions

- void ∗ gamelogic (void ∗p2GameState)

  *Thread principal donde se maneja la logica del videojuego.*
- char wasLevelInitialized ()

  *Checkea si el nivel esta inicializado.*
- sem_t ∗ getPhysicsSem ()

  *Getter para semaforo para fisicas.*
- sem_t ∗ getAnimeSem ()

  *Getter para semaforo para animaciones.*
- sem_t ∗ getRenderSem ()

  *Getter para semaforo para render.*

### 5.9.1 Function Documentation

#### 5.9.1.1 gamelogic()

```
void* gamelogic (
            void * p2GameState )
```

**Parameters**

| | |
|---|---|
| *p2GameState* | puntero al estado del juego |

#### 5.9.1.2 getAnimeSem()

```
sem_t* getAnimeSem ( )
```

**Returns**

Devuelve un puntero al semaforo para el thread de animaciones

#### 5.9.1.3 getPhysicsSem()

```
sem_t* getPhysicsSem ( )
```

**Returns**

Devuelve un puntero al semaforo para el thread de fisicas

#### 5.9.1.4 getRenderSem()

```
sem_t* getRenderSem ( )
```

**Returns**

Devuelve un puntero al semaforo para el thread de render

#### 5.9.1.5 wasLevelInitialized()

```
char wasLevelInitialized ( )
```

**Returns**

Devuelve 1 si esta inicializado y 0 en el caso contrario

## 5.10 IEvents.h File Reference

Interpretacion de eventos de teclado y joystick.

### Enumerations

- enum {
  **NOHAYMOVIMIENTO** = 50 , **UPDERECHA** , **DOWNDERECHA** , **UPABAJODERECHA** ,
  **DOWNABAJODERECHA** , **UPABAJO** , **DOWNABAJO** , **UPABAJOIZQUIERDA** ,
  **DOWNABAJOIZQUIERDA** , **UPIZQUIERDA** , **DOWNIZQUIERDA** , **UPARRIBAIZQUIERDA** ,
  **DOWNARRIBAIZQUIERDA** , **UPARRIBA** , **DOWNARRIBA** , **UPARRIBADERECHA** ,
  **DOWNARRIBADERECHA** , **UPBOTON** , **DOWNBOTON** , **DOWNESCAPE** ,
  **UPESCAPE** , **NUMOFEVENTKEYS** }

- enum {
  **DOWNA** = NUMOFEVENTKEYS , **UPA** , **DOWNB** , **UPB** ,
  **DOWNC** , **UPC** , **DOWND** , **UPD** ,
  **DOWNE** , **UPE** , **DOWNF** , **UPF** ,
  **DOWNG** , **UPG** , **DOWNH** , **UPH** ,
  **DOWNI** , **UPI** , **DOWNJ** , **UPJ** ,
  **DOWNK** , **UPK** , **DOWNL** , **UPL** ,
  **DOWNM** , **UPM** , **DOWNN** , **UPN** ,
  **DOWNO** , **UPO** , **DOWNP** , **UPP** ,
  **DOWNQ** , **UPQ** , **DOWNR** , **UPR** ,
  **DOWNS** , **UPS** , **DOWNT** , **UPT** ,
  **DOWNU** , **UPU** , **DOWNV** , **UPV** ,
  **DOWNW** , **UPW** , **DOWNX** , **UPX** ,
  **DOWNY** , **UPY** , **DOWNZ** , **UPZ** ,
  **DOWN0** , **UP0** , **DOWN1** , **UP1** ,
  **DOWN2** , **UP2** , **DOWN3** , **UP3** ,
  **DOWN4** , **UP4** , **DOWN5** , **UP5** ,
  **DOWN6** , **UP6** , **DOWN7** , **UP7** ,
  **DOWN8** , **UP8** , **DOWN9** , **UP9** ,
  **DOWNENTER** , **UPENTER** , **DOWNBACKSPACE** , **UPBACKSPACE** }

## Functions

- void ∗ keyboardChanges (void ∗myGameState)

  *Esta es la funcion que hace al thread que registra todos los eventos de teclado y los guarda en el buffer.*
- void ∗ InputEvent (void ∗gs)

  *Esta es la funcion que hace al thread que registra todos los eventos del joystick y los guarda en el buffer.*
- unsigned char getInputEvent (void)

  *Lee el siguiente elemento del buffer.*
- void storeInputEvent (unsigned char evento)

  *Se encarga de guardar los eventos que los threads leen en el buffer.*
- int esBufferVacio (void)

  *Checkea si el buffer esta vacio.*
- void limpiarBuffer (void)

  *Limpia el buffer de eventos.*

### 5.10.1 Function Documentation

#### 5.10.1.1 esBufferVacio()

```
int esBufferVacio (
            void  )
```

**Returns**

Devuelve 1 en el caso de que el buffer este vacio, y 0 en el caso contrario

#### 5.10.1.2 getInputEvent()

```
unsigned char getInputEvent (
            void  )
```

**Returns**

Devuelve el evento leido del buffer, el cual sera alguna de las constantes ya definidas en este archivo

#### 5.10.1.3 InputEvent()

```
void* InputEvent (
            void * gs )
```

**Parameters**

| | |
|---|---|
| *gs* | Un puntero a la estructura gameState |

**Returns**

Puntero a void que no se utiliza

#### 5.10.1.4 keyboardChanges()

```
void* keyboardChanges (
            void * myGameState )
```

**Parameters**

| | |
|---|---|
| *Un* | puntero a la estructura gameState |

**Returns**

puntero a void que no se utiliza

#### 5.10.1.5 storeInputEvent()

```
void storeInputEvent (
            unsigned char evento )
```

**Parameters**

| | |
|---|---|
| *evento* | El evento a guardar en el buffer, el cual sera alguna de las constantes ya definidas en este archivo |

## 5.11 joydrv.h File Reference

Joystick Driver.

### Classes

- struct jcoord_t

### Enumerations

- enum **jswitch_t** { **J_NOPRESS** , **J_PRESS** }

### Functions

- void joy_init (void)

  *Inicializa el joystick. Debe llamarse UNA vez antes de empezar a usar el joystick.*
- int joy_update (void)

  *Actualiza los valores medidos del joystick. Por lo tanto, debe llamarse antes de usar joy_get_coord() o joy_get_switch()*
- jcoord_t joy_get_coord (void)

  *Devuelve las coordenadas del joystick medidas al momento del último joy_update()*
- jswitch_t joy_get_switch (void)

  *Es análogo a joy_get_coord, en este caso devuelve el estado del switch del joystick.*

### 5.11.1 Detailed Description

**Author**

Daniel Jacoby, Nicolas Magliola y Pablo Vacatello

### 5.11.2 Function Documentation

#### 5.11.2.1 joy_get_coord()

```
jcoord_t joy_get_coord (
            void  )
```

**Returns**

Una estructura de tipo jcoord_t con las coordenadas del joystick.

#### 5.11.2.2 joy_get_switch()

```
jswitch_t joy_get_switch (
            void  )
```

**Returns**

Una variable de tipo jswich_t, es decir, J_NOPRESS o J_PRESS.

## 5.12 level.h File Reference

Funciones de inicializacion, manejo y dibujado de aquello relacionado al nivel.

### Functions

- int cargarMapa (level_t *level, int id)

  *Se detalla en memoria dinamica cual es cada uno de los bloques que hay en el nivel especificado.*
- void destroyMap (estadoJuego_t *gameState)

  *Libera el espacio en memoria reservado dinamicamente por cargarMapa.*
- void drawLevel (estadoJuego_t *gameState)

  *A partir de los bloques cargados anteriormente se encarga de dibujar el nivel, en allegro y en la raspi.*
- void initUI (gameUI_t *gameUI)

  *Se ponen los valores iniciales del tiempo, las monedas, el puntaje y el primer nivel.*
- int initEntities (estadoJuego_t *gameState)

  *Tomando la informacion de cargarmapa, se inicializa bloques y enemigos con su respectiva posicion, velocidad, tamaño. Tambien se hace un backup de esta info.*
- void drawRetryScreen (estadoJuego_t *gameState)

  *Dibuja las vidas restantes en pantalla en caso de perder una. Esto se realiza en modo allegro y tambien en raspi.*
- void drawGameOverScreen (estadoJuego_t *gameState)

  *Se avisa al jugador que perdio y se muestra su puntaje final. Luego se avisa en el caso de haberse realizado un HIGH SCORE.*
- void drawPause (estadoJuego_t *gameState)

  *Dibuja el menu de pausa. Esta funcion solo esta disponible para el modo allegro, para la raspi es una funcion nula.*
- void destroyEntities (estadoJuego_t *gameState)

  *Se libera el espacio reservado para los bloques y los enemigos, tanto en el original como el backup.*
- void resetEntitiesState (estadoJuego_t *gameState)

  *Se reinician todas los bloques y enemigos ya inicializados en el juego mediante el uso del backup creado por initBackupEntities Funcion privada que es llamada por initEntities.*
- void drawNextLevelScreen (estadoJuego_t *gameState)

  *En el caso de ganar el nivel, se avisa y se dibuja el score actual. Para modo Allegro tambien se dibujan las vidas.*
- void resetWavePosition (void)

  *Pone la posicion de la ola en 0.*
- int wasNewHighScoreAchieved (estadoJuego_t *gameState)

  *Se compara el score actual con la tabla de highscores para saber si se realizo un nuevo highscore.*
- void saveNewHighScore (estadoJuego_t *gameState)

  *Se guarda el nuevo highscore en su respectiva posicion en el archivo .txt que los contiene.*
- void drawGameOverScreenHighScore (estadoJuego_t *gameState)

  *En el caso de haberse realizado un high score, esta funcion lo avisa en el modo RASPI.*

### 5.12.1 Function Documentation

#### 5.12.1.1 cargarMapa()

```
int cargarMapa (
            level_t * level,
            int id )
```

**Parameters**

| | |
|---|---|
| *level* | puntero a la estructura level_t donde se reservara memoria dinamica y se guardaran todos los bloques |
| *id* | aqui se expecifica el nivel del cual se debera cargar el mapa |

**Returns**

Devuelve 0 si no hubo ningun error y 1 si hubo alguno

#### 5.12.1.2 destroyEntities()

```
void destroyEntities (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t de donde se obtienen los punteros para liberarlos |

#### 5.12.1.3 destroyMap()

```
void destroyMap (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | Puntero al estado del juego desde donde se accede a todos los bloques |

#### 5.12.1.4 drawGameOverScreen()

```
void drawGameOverScreen (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t donde obtiene la informacion para realizar el dibujado |

#### 5.12.1.5 drawGameOverScreenHighScore()

```
void drawGameOverScreenHighScore (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t donde obtiene la informacion para realizar el dibujado |

#### 5.12.1.6 drawLevel()

```
void drawLevel (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *∗gameState* | puntero al estado del juego del cual consigue la posicion y las imagenes de aquello que hay que dibujar |

#### 5.12.1.7 drawNextLevelScreen()

```
void drawNextLevelScreen (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t donde obtiene la informacion para realizar el dibujado |

#### 5.12.1.8 drawPause()

```
void drawPause (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t donde obtiene la informacion para realizar el dibujado |

#### 5.12.1.9 drawRetryScreen()

```
void drawRetryScreen (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t donde obtiene la informacion para realizar el dibujado |

#### 5.12.1.10 initEntities()

```
int initEntities (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | Puntero a estadoJuego_t, donde se toma la informacion dicha anteriormente y luego se guarda lo nuevo |

**Returns**

Devuelve 0 si no hubo ningun error y 1 si hubo alguno

#### 5.12.1.11 initUI()

```
void initUI (
            gameUI_t * gameUI )
```

**Parameters**

| | |
|---|---|
| *gameUI* | puntero a gameUI_t estructura donde se escribe lo anterior |

#### 5.12.1.12 resetEntitiesState()

```
void resetEntitiesState (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *puntero* | a estadoJuego_t de donde se obtiene el estado actual y el default del juego |

#### 5.12.1.13 saveNewHighScore()

```
void saveNewHighScore (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t del cual se obtienen los highscores |

#### 5.12.1.14 wasNewHighScoreAchieved()

```
int wasNewHighScoreAchieved (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *gameState* | puntero a estadoJuego_t del cual se obtienen los highscores |

**Returns**

Devuelve 0 en el caso de no ser highscore y 1 en el caso de serlo

## 5.13 main.c File Reference

### Functions

- int **main** (void)

## 5.14 matiasBrosGame.h File Reference

Header que contiene algunos estados del juego y donde principalmente se define estadoJuego_t.

### Classes

- struct bufferRecursos_t
- struct level_t
- struct gameUI_t
- struct estadoJuego_t

### Typedefs

- typedef Audio * **sonido_t**
- typedef enum ESTADOSDELJUEGO **estadosjuego_t**
- typedef ALLEGRO_BITMAP * **image_t**
- typedef ALLEGRO_FONT * **fuente_t**

### Enumerations

- enum **POWERUPSSTATE** { **SMALL** , **BIG** }
- enum **ESTADOSDELJUEGO** {
  **MENU** = 10 , **CHOOSINGLEVEL** , **INSCORETABLE** , **INGAME** ,
  **RETRYSCREEN** , **GAMECLOSED** , **PAUSE** , **NEXTLEVEL** ,
  **GAMEOVERSCREEN** }
- enum **MENUOPTIONS** { **LEVELSELECTOR** = 1 , **SCORETABLE** , **EXITGAME** }
- enum **PAUSEOPTIONS** { **RESUME** = 0 , **BACKTOMENU** }

## 5.15 menu.h File Reference

Funciones relacionadas al dibujado y manejo del menu.

### Functions

- void destroyMenu (void)

  *Libera el espacio reservado en memoria para las imagenes y texto del menu.*
- void imprimirNumero (int numero, int zeroEnabled)

  *Dibuja el numero recibido en la parte inferior del display de la raspi. 4 cifras maximo.*
- void drawMenu (estadoJuego_t *gameState)

  *Dibuja el menu.*
- int loadMenuData (void)

  *Carga la informacion de los textos y las imagenes del menu dentro del juego (en memoria dinamica) para luego poder dibujar tod0. Para la Raspi esta funcion no hace nada.*
- void drawLevelSelector (estadoJuego_t *gameState)

  *Dibuja la pantalla de seleccion de nivel.*
- void drawTopScores (estadoJuego_t *gameState)

  *Dibuja la pantalla de highscores.*
- void updateMenuArrow (int *seleccion, unsigned char evento)

  *Actualiza la flecha del menu segun los limites de la misma.*
- void updatePauseArrow (int *seleccion, unsigned char evento)

  *Actualiza la flecha del menu de pausa segun los limites de la misma.*

### 5.15.1 Function Documentation

#### 5.15.1.1 drawLevelSelector()

```
void drawLevelSelector (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *∗gameState* | puntero a gameState donde se encuentra toda la informacion sobre las imagenes y el nivel seleccionado |

#### 5.15.1.2 drawMenu()

```
void drawMenu (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *∗gameState* | puntero a gameState donde se encuentra toda la informacion sobre las imagenes y texto del menu |

#### 5.15.1.3 drawTopScores()

```
void drawTopScores (
            estadoJuego_t * gameState )
```

**Parameters**

| | |
|---|---|
| *∗gameState* | puntero a gameState donde se encuentra toda la informacion sobre las imagenes y texto de la pantalla de highscores |

#### 5.15.1.4 imprimirNumero()

```
void imprimirNumero (
            int numero,
            int zeroEnabled )
```

**Parameters**

| | |
|---|---|
| *numero* | es el numero a dibujar |
| *zeroEnabled* | Si se indica 1, entonces se imprimen las 4 cifras. Si se indica 0, entonces se imprime una sola a la derecha |

#### 5.15.1.5 loadMenuData()

```
int loadMenuData (
            void  )
```

**Returns**

Devuelve 0 si la operacion fue correcta y 1 en el caso contrario

#### 5.15.1.6 updateMenuArrow()

```
void updateMenuArrow (
            int * seleccion,
            unsigned char evento )
```

**Parameters**

| | |
|---|---|
| ∗*seleccion* | recibe un puntero a la posicion donde debe modificar la seleccion |
| *evento* | Recibe el evento realizado, es decir arriba, abajo o izquierda-derecha |

#### 5.15.1.7 updatePauseArrow()

```
void updatePauseArrow (
            int * seleccion,
            unsigned char evento )
```

**Parameters**

| | |
|---|---|
| ∗*seleccion* | recibe un puntero a la posicion donde debe modificar la seleccion |
| *evento* | Recibe el evento realizado, es decir arriba, abajo o izquierda-derecha |

## 5.16 raspi.h File Reference

Header que contiene todos las librerias de la raspi.

## 5.17 render.h File Reference

Controla el refresco de pantalla y lo que se muestra en ella.

### Functions

- void * render (void *gs)

  *Se encarga de refrescar la pantalla cada cierto tiempo indicado por timer.*
- void updateCameraPosition (void *gs)

  *Actualiza la posición de la pantalla sobre el nivel.*
- int isInsideScreenX (fisica_t *object1)

  *Determina si cierta entidad se encuentra o no en la pantalla (eje X)*
- void setCameraScrollX (float coordX)

  *Asigna el primer parámetro a la posición actual de la pantalla en el nivel.*
- float getCameraScrollX ()

  *Devuelve la posición actual de la pantalla en el nivel.*
- void resetLastBlockInMap ()

  *Asigna NULL a la variable LastBlockInMap.*
- void writeDisplay (char matriz[16][16])

  *Esta función controla el buffer del display a partir de la matriz dada, si un elemento vale 0 se apaga el led correspondiente, si es 1 se prende, no hay cambios en cualquier otro caso.*
- int isInsideScreenY (fisica_t *object1)

  *Determina si cierta entidad se encuentra o no en la pantalla (eje Y)*

### 5.17.1 Function Documentation

#### 5.17.1.1 getCameraScrollX()

```
float getCameraScrollX ( )
```

**Returns**

Devuelve el valor correspondiente en tipo flotante

#### 5.17.1.2 isInsideScreenX()

```
int isInsideScreenX (
            fisica_t * object1 )
```

**Parameters**

| | |
|---|---|
| *object1* | puntero a estructura fisica_t de la entidad a analizar |

**Returns**

Devuelve 1 si la entidad se encuentra en la pantalla, en caso contrario devuelve 0

#### 5.17.1.3 isInsideScreenY()

```
int isInsideScreenY (
            fisica_t * object1 )
```

**Parameters**

| | |
|---|---|
| *object1* | puntero a estructura fisica_t de la entidad a analizar |

**Returns**

Devuelve 1 si la entidad se encuentra en la pantalla, en caso contrario devuelve 0

#### 5.17.1.4 render()

```
void* render (
            void * gs )
```

**Parameters**

| | |
|---|---|
| *gs* | Se debe pasar un puntero a la estructura principal del programa |

#### 5.17.1.5 setCameraScrollX()

```
void setCameraScrollX (
            float coordX )
```

**Parameters**

| | |
|---|---|
| *coordX* | Valor que se desea asignar |

#### 5.17.1.6 updateCameraPosition()

```
void updateCameraPosition (
            void * gs )
```

**Parameters**

| | |
|---|---|
| *gs* | Se debe pasar un puntero a la estructura principal del programa |

#### 5.17.1.7 writeDisplay()

```
void writeDisplay (
            char matriz[16][16] )
```

**Parameters**

| | |
|---|---|
| *matriz* | Matriz con datos para el control de leds |

## 5.18 times.h File Reference

Libreria de timer.

### Classes

- struct timer

### Typedefs

- typedef struct timer **eventTimer_t**

### Enumerations

- enum {
  **FPSTIMER** , **INGAMETIMER** , **PHYSICSTIMER** , **ANIMETIMER** ,
  **DOVULNERABLETIMER** , **DEATHANIM** , **PLAYERSWIMMINGANIM** , **SEAWEEDANIM** ,
  **BLINKINGCOINANIM** , **CHEEPCHEEPANIM** , **LIFEUPANIM** , **MUSHROOMANIM** ,
  **PIPEANIM** , **NUMOFDEFAULTTIMERS** }

## Functions

- void setCurrentGameState (void *gs)

  *La funcion recibe el puntero al gameState del juego para que lo usen los enemigos.*
- int createNewTimer (float _secondsPerTick, void(*funct)(void *), int ID)

  *Reserva espacio para una nueva estructura eventTimer_t que dispara cada _secondsPerTick la funcion funct y se guarda con el ID ingresado. En caso de que el ID ya exista, no hace nada.*
- void startTimer (int timerID)

  *Comienza a correr el timer con el ID que se le paso por parametro. Si el timer no existe, no hace nada.*
- void stopTimer (int timerID)

  *Para de correr el timer con el ID que se le paso por parametro. Si el timer no existe, no hace nada.*
- int isPaused (int timerID)

  *La funcion indica el estado de pausa del reloj con ID timerID.*
- void destroyAllTimers ()

  *Libera la memoria de todos los timers que se crearon.*

### 5.18.1 Function Documentation

#### 5.18.1.1 createNewTimer()

```
int createNewTimer (
            float _secondsPerTick,
            void(*)(void *) funct,
            int ID )
```

**Parameters**

| | |
|---|---|
| *_secondsPerTick* | cantidad de segundos que deben transcurrir para que el reloj dispare la funcion pasada por parametro |
| *funct* | puntero a la funcion que se disparara |
| *ID* | identificador unico para acceder al reloj |

**Returns**

Devuelve un 0 si se pudo crear el reloj exitosamente, sino devuelve un -1

#### 5.18.1.2 isPaused()

```
int isPaused (
            int timerID )
```

**Parameters**

| | |
|---|---|
| *timerID* | identificador del timer que se quiere evaluar. |

**Returns**

Devuelve 0 si el reloj NO esta pausado o no existe el reloj, 1 si esta pausado

#### 5.18.1.3 setCurrentGameState()

```
void setCurrentGameState (
            void * gs )
```

**Parameters**

| | |
|---|---|
| *Puntero* | al estadoJuego del juego |

#### 5.18.1.4 startTimer()

```
void startTimer (
            int timerID )
```

**Parameters**

| | |
|---|---|
| *timerID* | identificador del timer que se quiere lanzar |

#### 5.18.1.5 stopTimer()

```
void stopTimer (
            int timerID )
```

**Parameters**

| | |
|---|---|
| *timerID* | identificador del timer que se quiere parar. |

# Index