MISSILE COMMAND PROJECT

API Documentation

Gatech

FALL 2016 ECE2035

Table of Contents

city_landscape_public.h File Reference

Functions

void city_landscape_init (int num_city) Call city_landscape_init() only once at the begining of your code.
CITY city_get_info (int index) Get the information of city.
void city_destory (int index) Remove the city from record and screen.
void draw_cities (void) Draw all exist cities onto the screen.
void draw_landscape (void) Draw the landscape.

Detailed Description

Definition in file city_landscape_public.h.

Enumeration Type Documentation

enum CITY_STATUS

The enum define the status of a city.
Enumerator:

EXIST	The city will be shown on screen.
DESTORIED	The city won't be shown on screen.

Definition at line 11 of file city_landscape_public.h.

Function Documentation

void city_destory (int *index*)

Remove the city from record and screen.
Parameters:

index The index in city_record. It must be smaller than MAX_NUM_CITY.

Definition at line 54 of file city_landscape.cpp.

CITY city_get_info (int *index*)

Get the information of city.
Parameters:

index The index in city_record. It must be smaller than MAX_NUM_CITY.

Returns:

The structure of city information

Definition at line 47 of file city_landscape.cpp.

void city_landscape_init (int *num_city*)

Call city_landscape_init() only once at the begining of your code.
Parameters:

num_city number of city to be draw. It must be less/equal to MAX_NUM_CITY.

Definition at line 13 of file city_landscape.cpp.

void draw_cities (void)

Draw all exist cities onto the screen.
You might not need to use this function, but you could still use it if you want.
Definition at line 76 of file city_landscape.cpp.

void draw_landscape (void)

Draw the landscape.
You might not need to use this function, but you could still use it if you want.
Definition at line 99 of file city_landscape.cpp.

missile_public.h File Reference

Function Documentation

DLinkedList* get_missile_list ( )

This function will return a linked-list of all active MISSILE structures.
This can be used to modify the active missiles. Marking missiles with status MISSILE_EXPLODED will cue their erasure from the screen and removal from the list at the next missile_generator() call.
Definition at line 104 of file missile.cpp.

void missile_generator (void)

This function draw the missiles onto the screen Call missile_generator() repeatedly in your game-loop.
ex: main()
Definition at line 21 of file missile.cpp.

void missile_init (void)

Call missile_init() only once at the begining of your code.
Definition at line 15 of file missile.cpp.

void set_missile_interval (int *interval*)

Set the interval that the missiles fire, interval has range of 1-100 with 1 being fired in very quick succession and 100 being fired very slowly after one another.
Definition at line 96 of file missile.cpp.

void set_missile_speed (int *speed*)

Set the speed of missiles, Speed has range of 1-8 with 1 being fastest and 8 being slowest.
Definition at line 88 of file missile.cpp.

player_public.h File Reference

enum PLAYER_STATUS { ALIVE = 1, DESTROYED = 0 }

The enum define the status of a player.

Functions

PLAYER player_get_info (void) Get the information of player.
void player_init (void) Initialize the player's attributes,including position, missile status.
void player_moveLeft (void) Move delta pixels to the left.
void player_moveRight (void) Move delta pixels to the right.
void player_fire (void) Fire missiles.
void player_missile_draw (void) Updates the drawing of missiles on screen.
void player_destroy (void) Destroy the player to end game.

Detailed Description

Definition in file player_public.h.

Enumeration Type Documentation

enum PLAYER_MISSILE_STATUS

The enum define the status of a player missile.
Enumerator:

PMISSILE_EXPLODED	The PMISSILE is deactive.
PMISSILE_ACTIVE	The PMISSILE is active.

Definition at line 14 of file player_public.h.

enum PLAYER_STATUS

The enum define the status of a player.
Enumerator:

ALIVE	The player is alive.
DESTROYED	The player is dead.

Definition at line 27 of file player_public.h.

Function Documentation

void player_destroy (void)

Destroy the player to end game.
Definition at line 73 of file player.cpp.

void player_fire (void)

Fire missiles.
Definition at line 30 of file player.cpp.

PLAYER player_get_info (void)

Get the information of player.
Returns:

The structure of player information

Definition at line 5 of file player.cpp.

void player_init (void)

Initialize the player's attributes,including position, missile status.
Also, draw the player
Definition at line 10 of file player.cpp.

void player_missile_draw (void)

Updates the drawing of missiles on screen.
Definition at line 35 of file player.cpp.

void player_moveLeft (void)

Move delta pixels to the left.
Definition at line 20 of file player.cpp.

void player_moveRight (void)

Move delta pixels to the right.
Definition at line 25 of file player.cpp.

doubly_linked_list.h File Reference

Typedef Documentation

typedef struct dlinkedlist_t DLinkedList

The structure to store the information of a doubly linked list.

typedef struct llnode_t LLNode

The structure to store the information of a doubly linked list node.

Function Documentation

DLinkedList* create_dlinkedlist (void)

create_dlinkedlist
Creates a doublely liked list by allocating memory for it on the heap. Initialize the size to zero, as well as head, current, and tail pointer to NULL

Returns:

A pointer to an empty dlinkedlist

Definition at line 31 of file doubly_linked_list.cpp.

void* deleteBackward (DLinkedList * *dLinkedList*, int *shouldFree*)

deleteBackward
Delete the node the current pointer is pointed at, and move the current pointer backwards. Be aware that deleteBackward will cause problem if the current pointer is pointing at list head

Parameters:

dLinkedList	A pointer to the doubly linked list
shouldFree	Flag. 1 indicates if data should be freed upon deletion of node.

Returns:

the data of the new current pointer and NULL if the current pointer is NULL

Definition at line 68 of file doubly_linked_list.cpp.

void* deleteForward (DLinkedList * *dLinkedList*, int *shouldFree*)

deleteForward
Delete the node the current pointer is pointed at, and move the current pointer forwards. Be aware that deleteForward will cause problem if the current pointer is pointing at list tail

Parameters:

dLinkedList	A pointer to the doubly linked list
shouldFree	Flag. 1 indicates if data should be freed upon deletion of node.

Returns:

the data of the new current pointer and NULL if the current pointer is NULL

Definition at line 72 of file doubly_linked_list.cpp.

void destroyList (DLinkedList * *dLinkedList*, int *shouldFree*)

destroyList
Destroy the doublely linked list. Everything in the linked list including list structure, nodes and data are all freed from the heap

Parameters:

dLinkedList	A pointer to the doubly linked list
shouldFree	Flag. 1 indicates if data should be freed upon deletion of node.

Definition at line 76 of file doubly_linked_list.cpp.

void* getCurrent (DLinkedList * *dLinkedList*)

getCurrent

Return the data the current pointer is pointing at
Parameters:

dLinkedList A pointer to the doubly linked list

Returns:
the current data or NULL if current == NULL
Definition at line 97 of file doubly_linked_list.cpp.

void* getHead (DLinkedList * *dLinkedList*)

getHead
Return the data contained in the head of the doublely linked list, and set the list current pointer to head
Parameters:

dLinkedList A pointer to the doubly linked list

Returns:
the head data or NULL if head == NULL
Definition at line 84 of file doubly_linked_list.cpp.

void* getNext (DLinkedList * *dLinkedList*)

getNext
Return the next data the current pointer is pointing at, and move the current pointer to next node
Parameters:

dLinkedList A pointer to the doubly linked list

Returns:
the next data or NULL if current == NULL
Definition at line 101 of file doubly_linked_list.cpp.

void* getPrevious (DLinkedList * *dLinkedList*)

getPrevious
Return the previous data the current pointer is pointing at, and move the current pointer to previous node
Parameters:

dLinkedList A pointer to the doubly linked list

Returns:
the previous data or NULL if current == NULL
Definition at line 105 of file doubly_linked_list.cpp.

int getSize (DLinkedList * *dLinkedList*)

getSize
Return the size of the doubly linked list
Parameters:

dLinkedList A pointer to the doubly linked list

Returns:
the size
Definition at line 109 of file doubly_linked_list.cpp.

void* getTail (DLinkedList * *dLinkedList*)

getTail
Return the data contained in the tail of the doublely linked list, and set the current pointer to tail
Parameters:

dLinkedList A pointer to the doubly linked list

Returns:
the tail data or NULL if tail == NULL
Definition at line 93 of file doubly_linked_list.cpp.

(DLinkedList * ***dLinkedList*** ***,***
void * ***newData***
)

insertAfter
Insert the new data to the doublely linked list right after the current pointer
Parameters:

dLinkedList A pointer to the doubly linked list

newData A void pointer to the new data that the user want to add after data

Returns:

1 if insert the new data successfully 0 if the current pointer is NULL

Definition at line 60 of file doubly_linked_list.cpp.

int insertBefore (DLinkedList * *dLinkedList*,
void * *newData*
)

insertBefore

Insert the new data to the doublely linked list right before the current pointer

Parameters:

dLinkedList	A pointer to the doubly linked list
newData	A void pointer to the new data that the user want to add after data

Returns:

1 if insert the new data successfully 0 if the current pointer is NULL

Definition at line 64 of file doubly_linked_list.cpp.

void insertHead (DLinkedList * *dLinkedList*,
void * *data*
)

InsertHead.

Insert the data to the head of the doublely linked list. Update the current pointer of the list only when it is originally NULL.

Parameters:

dLinkedList	A pointer to the doubly linked list
data	A void pointer to data the user is adding to the doublely linked list.

Definition at line 40 of file doubly_linked_list.cpp.

void insertTail (DLinkedList * *dLinkedList*,
void * *data*
)

insertTail

Insert the data to the tail of the doublely linked list. Update the current pointer of the list only when it is originally NULL

Parameters:

dLinkedList	A pointer to the doubly linked list
data	A void pointer to data the user is adding to the doublely linked list.

Definition at line 56 of file doubly_linked_list.cpp.