**Файл «main.c»**

#include <ncurses.h>

#include <unistd.h>

#include "console.h"

#include <locale.h>

#include "sprite.h"

#include "logger.h"

#include "sprites.h"

#include "map.h"

#include <time.h>

#include "bomberman.h"

#include "config.h"

#include "menu.h"

int main()

{

srand(time(0));

setlocale(LC\_ALL, "");

initscr();

if (COLS < 60 || LINES < 26) {

endwin();

printf("Window is too small\n");

return -1;

}

noecho();

cbreak();

keypad(stdscr, TRUE);

nodelay(stdscr,TRUE);

curs\_set(0);

scrollok(stdscr, FALSE);

Logger\_init();

Console\_init();

Sprites\_init();

Config\_init();

Config\_loadFile("config");

Menu\_run();

Sprites\_free();

endwin();

return 0;

}

**Файл «game.h»**

#ifndef H\_GAME

#define H\_GAME

#include "map.h"

#include "logger.h"

#include "bomberman.h"

#include "config.h"

#include "sprites.h"

#include <ncurses.h>

void Game\_run(int playerCount);

#endif

**Файл «game.c»**

#include "game.h"

void Game\_run(int playerCount)

{

struct Map map;

Map\_init(&map, 15, 13, playerCount);

int timeBeforeQuit = 200;

int done = 0;

while(!done){

Console\_clear();

int key = getch();

if(key!=ERR)

{

LOG("You pressed %d",key);

Map\_handleKey(&map, key);

if(key==27)

{

done = 1;

}

}

Map\_update(&map);

int count = 0;

int i;

for(i=0;i<4;i++)

{

if(map.bombermans[i].alive)

{

count = count + 1;

}

}

if(count<=1)

{

timeBeforeQuit = timeBeforeQuit - 1;

if(timeBeforeQuit<=0)

done = 1;

}

Map\_draw(&map);

Console\_draw();

usleep(1000000/30);

}

Map\_free(&map);

}

**Файл «menu.h»**

#ifndef H\_MENU

#define H\_MENU

#include "logger.h"

#include "sprites.h"

#include "console.h"

void Menu\_run();

#endif

**Файл «menu.c»**

#include "menu.h"

#include "game.h"

void Menu\_run()

{

int selected = 0;

int playerCount = 1;

int time = 0;

int done = 0;

while(!done){

Console\_clear();

int key = getch();

if(key!=ERR)

{

LOG("You pressed %d",key);

if(key==119 || key==259)

{

selected = selected + 1;

if(selected>=2)

selected = 0;

}

if(key==115 || key==258)

{

selected = selected - 1;

if(selected<0)

selected = 1;

}

if(selected==0 && (key==97 || key==260))

{

playerCount = playerCount - 1;

if(playerCount<=0)

playerCount = 4;

}

if(selected==0 && (key==100 || key==261))

{

playerCount = playerCount + 1;

if(playerCount>4)

playerCount = 1;

}

if(key==27 || (selected==1 && (key==101 || key==32 || key==10)))

{

done = 1;

}

if(selected==0 && (key==101 || key==32 || key==10))

{

Game\_run(playerCount);

}

}

Sprite\_draw(Sprites\_get(S\_LOGO),0,2);

Sprite\_draw(Sprites\_get(S\_COPYRIGHT),0,24);

if(time>=15)

{

Sprite\_draw(Sprites\_get(S\_MENUBOMB),45,19);

}else

{

Sprite\_draw(Sprites\_get(S\_MENUBOMB2),45,19);

}

if(playerCount==1)

Console\_drawStr(26,12,L"1 Player");

else if(playerCount==2)

Console\_drawStr(26,12,L"2 Players");

else if(playerCount==3)

Console\_drawStr(26,12,L"3 Players");

else if(playerCount==4)

Console\_drawStr(26,12,L"4 Players");

Console\_drawStr(28,14,L"Quit");

time = time + 1;

if(time>50)

time = 0;

wchar\_t ch = L'▶';

wchar\_t ch2 = L'◀';

if(time>=25)

{

ch = L'▷';

ch2 = L'◁';

}

if(selected==0)

{

Console\_drawChar(24,12,ch2);

Console\_drawChar(36-(playerCount==1),12,ch);

}else

{

Console\_drawChar(26,14,ch);

}

Console\_draw();

usleep(1000000/30);

}

}

**Файл «config.h»**

#ifndef H\_CONFIG

#define H\_CONFIG

struct PlayerControls

{

int up;

int down;

int left;

int right;

int bomb;

};

struct Config

{

struct PlayerControls controls[4];

int bombTimer;

int bombFireTimer;

int bombFireCenterTimer;

int rockChance;

};

struct Config\* Config\_get();

void Config\_init();

void Config\_loadFile(char\* path);

#endif

**Файл «config.c»**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include "logger.h"

struct Config\* Config\_get()

{

static struct Config config;

return &config;

}

void Config\_init()

{

struct Config\* self = Config\_get();

self->bombTimer = 60;

self->bombFireTimer = 50;

self->bombFireCenterTimer = 60;

self->rockChance = 80;

}

void Config\_loadFile(char\* path)

{

struct Config\* self = Config\_get();

FILE \*fp = fopen(path, "r");

if(fp==0)

{

return;

}

while(!feof(fp))

{

char com[255];

fscanf(fp, "%s", com);

if(strcmp(com,"controls") == 0)

{

int playerid;

fscanf(fp, "%d", &playerid);

char dir[10];

fscanf(fp, "%s", dir);

int key;

fscanf(fp, "%d", &key);

if(strcmp(dir,"up") == 0)

{

self->controls[playerid].up = key;

LOG("Binded key %d to up of player %d",key,playerid);

}

if(strcmp(dir,"down") == 0)

{

self->controls[playerid].down = key;

LOG("Binded key %d to down of player %d",key,playerid);

}

if(strcmp(dir,"left") == 0)

{

self->controls[playerid].left = key;

LOG("Binded key %d to left of player %d",key,playerid);

}

if(strcmp(dir,"right") == 0)

{

self->controls[playerid].right = key;

LOG("Binded key %d to right of player %d",key,playerid);

}

if(strcmp(dir,"bomb") == 0)

{

self->controls[playerid].bomb = key;

LOG("Binded key %d to bomb of player %d",key,playerid);

}

}else

if(strcmp(com,"bombTimer") == 0)

{

int bombTimer;

fscanf(fp, "%d", &bombTimer);

self->bombTimer = bombTimer;

}else

if(strcmp(com,"bombFireTimer") == 0)

{

int bombFireTimer;

fscanf(fp, "%d", &bombFireTimer);

self->bombFireTimer = bombFireTimer;

}else

if(strcmp(com,"bombFireCenterTimer") == 0)

{

int bombFireCenterTimer;

fscanf(fp, "%d", &bombFireCenterTimer);

self->bombFireCenterTimer = bombFireCenterTimer;

}else

if(strcmp(com,"rockChance") == 0)

{

int rockChance;

fscanf(fp, "%d", &rockChance);

self->rockChance = rockChance;

}

}

}

**Файл «sprite.h»**

#ifndef H\_SPRITE

#define H\_SPRITE

#include <wchar.h>

struct Sprite

{

int w;

int h;

wchar\_t\* data;

};

struct Sprite\* Sprite\_init(int w, int h);

struct Sprite\* Sprite\_fromFile(char\* path);

void Sprite\_free(struct Sprite\* self);

void Sprite\_set(struct Sprite\* self, int x, int y, wchar\_t c);

wchar\_t Sprite\_get(struct Sprite\* self, int x, int y);

void Sprite\_draw(struct Sprite\* self, int x, int y);

#endif

**Файл «sprite.c»**

#include "sprite.h"

#include <stdlib.h>

#include <stdio.h>

#include "console.h"

#define TRANSPARENT\_CHARACTER L'~'

struct Sprite\* Sprite\_init(int w, int h)

{

struct Sprite\* self = (struct Sprite\*)malloc(sizeof(struct Sprite));

self->w = w;

self->h = h;

self->data = malloc(sizeof(wchar\_t)\*w\*h);

int i;

for(i = 0;i<w\*h;i++)

{

self->data[i] = TRANSPARENT\_CHARACTER;

}

return self;

}

void Sprite\_free(struct Sprite\* self)

{

free(self->data);

free(self);

}

struct Sprite\* Sprite\_fromFile(char\* path)

{

FILE \*fp = fopen(path, "r");

if(fp==0)

{

return 0;

}

int w,h;

fscanf(fp, "%d %d", &w, &h);

struct Sprite\* spr = Sprite\_init(w,h);

int i = 0;

while(i<w\*h)

{

wchar\_t c;

fscanf(fp, "%lc", &c);

if(c!=L'\n')

{

spr->data[i] = c;

i = i + 1;

}

}

fclose(fp);

return spr;

}

void Sprite\_set(struct Sprite\* self, int x, int y, wchar\_t c)

{

self->data[y\*self->w+x] = c;

}

wchar\_t Sprite\_get(struct Sprite\* self, int x, int y)

{

return self->data[y\*self->w+x];

}

void Sprite\_draw(struct Sprite\* self, int x, int y)

{

if(self==0)

return;

int xx;

int yy;

for(yy=0;yy<self->h;yy++)

{

for(xx=0;xx<self->w;xx++)

{

Console\_drawChar(xx+x,yy+y,Sprite\_get(self,xx,yy));

}

}

}

**Файл «sprites.h»**

#ifndef H\_SPRITES

#define H\_SPRITES

enum SPRITES { S\_BOMBERMAN, S\_WALL, S\_FLOOR, S\_BRICKS, S\_BOMB3, S\_BOMB2, S\_BOMB1, S\_FIRE, S\_GRAVE, S\_BONUS1, S\_BONUS2, S\_LOGO, S\_COPYRIGHT, S\_MENUBOMB, S\_MENUBOMB2 };

struct Sprite;

struct Sprites

{

struct Sprite\* sprites[16];

};

struct Sprites\* Sprites\_getObject();

void Sprites\_load(enum SPRITES name, char\* path);

void Sprites\_init();

void Sprites\_free();

struct Sprite\* Sprites\_get(enum SPRITES name);

#endif

**Файл «sprites.c»**

#include "sprites.h"

#include "sprite.h"

struct Sprites\* Sprites\_getObject()

{

static struct Sprites s;

return &s;

}

void Sprites\_load(enum SPRITES name, char\* path)

{

struct Sprites\* self = Sprites\_getObject();

self->sprites[(int)name] = Sprite\_fromFile(path);

}

void Sprites\_init()

{

struct Sprites\* self = Sprites\_getObject();

int i;

for(i=0;i<16;i++)

{

self->sprites[i] = 0;

}

Sprites\_load(S\_BOMBERMAN,"sprites/bomberman");

Sprites\_load(S\_WALL,"sprites/wall");

Sprites\_load(S\_FLOOR,"sprites/floor");

Sprites\_load(S\_BRICKS,"sprites/bricks");

Sprites\_load(S\_BOMB3,"sprites/bomb3");

Sprites\_load(S\_BOMB2,"sprites/bomb2");

Sprites\_load(S\_BOMB1,"sprites/bomb1");

Sprites\_load(S\_FIRE,"sprites/fire");

Sprites\_load(S\_GRAVE,"sprites/grave");

Sprites\_load(S\_BONUS1,"sprites/bonus1");

Sprites\_load(S\_BONUS2,"sprites/bonus2");

Sprites\_load(S\_LOGO,"sprites/logo");

Sprites\_load(S\_COPYRIGHT,"sprites/copyright");

Sprites\_load(S\_MENUBOMB,"sprites/menubomb");

Sprites\_load(S\_MENUBOMB2,"sprites/menubomb2");

}

void Sprites\_free()

{

struct Sprites\* self = Sprites\_getObject();

int i;

for(i=0;i<16;i++)

{

if(self->sprites[i])

{

Sprite\_free(self->sprites[i]);

}

}

}

struct Sprite\* Sprites\_get(enum SPRITES name)

{

struct Sprites\* self = Sprites\_getObject();

return self->sprites[(int)name];

}

**Файл «semaphore.h»**

#ifndef H\_SEMAPHORE

#define H\_SEMAPHORE

struct Semaphore

{

int semId;

};

struct Semaphore\* Semaphore\_init(int n);

void Semaphore\_free(struct Semaphore\* self);

void Semaphore\_grab(struct Semaphore\* self);

void Semaphore\_release(struct Semaphore\* self);

#endif

“semaphore.c”

#include "semaphore.h"

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/sem.h>

#include <stdlib.h>

struct Semaphore\* Semaphore\_init(int n)

{

struct Semaphore\* self = (struct Semaphore\*)malloc(sizeof(struct Semaphore));

self->semId = semget(IPC\_PRIVATE, 1, 0600|IPC\_CREAT);

struct sembuf op;

op.sem\_op = n;

op.sem\_flg = 0;

op.sem\_num = 0;

semop(self->semId, &op, 1);

return self;

}

void Semaphore\_free(struct Semaphore\* self)

{

semctl(self->semId, 0, IPC\_RMID, 0);

free(self);

}

void Semaphore\_grab(struct Semaphore\* self)

{

struct sembuf op;

op.sem\_op = -1;

op.sem\_flg = 0;

op.sem\_num = 0;

semop(self->semId, &op, 1);

}

void Semaphore\_release(struct Semaphore\* self)

{

struct sembuf op;

op.sem\_op = 1;

op.sem\_flg = 0;

op.sem\_num = 0;

semop(self->semId, &op, 1);

}

**Файл «map.h»**

#ifndef H\_MAP

#define H\_MAP

#include "bomb.h"

#include "semaphore.h"

#include "bomberman.h"

#define BOMB\_COUNT 16

struct MapItem

{

int type;

int burnTime;

int hasPlayer;

int hasBomb;

int hasBonus;

int blockBombs;

};

struct Map

{

int w;

int h;

struct MapItem\* map;

struct Bomb bombs[BOMB\_COUNT];

struct Semaphore\* bombsSem[BOMB\_COUNT];

struct Semaphore\* mapSem;

struct Bomberman bombermans[4];

};

void Map\_init(struct Map\* self, int w,int h, int playerCount);

void Map\_draw(struct Map\* self);

void Map\_update(struct Map\* self);

void Map\_handleKey(struct Map\* self, int key);

int Map\_hasWall(struct Map\* self, int x, int y);

int Map\_hasSomething(struct Map\* self, int x, int y);

int Map\_wallType(struct Map\* self, int x, int y);

int Map\_createBomb(struct Map\* self, int x,int y, struct Bomberman\* b);

struct MapItem\* Map\_get(struct Map\* self, int x,int y);

//ai

int Map\_safeDirection(struct Map\* self, int x,int y);

int Map\_inDanger(struct Map\* self, int x,int y);

int Map\_safeDistIfBomb(struct Map\* self, int px,int py, struct Bomberman\* b);

int Map\_dirToNearestEnemy(struct Map\* self, struct Bomberman\* b);

#endif

**Файл «map.c»**

#include "map.h"

#include "console.h"

#include <stdlib.h>

#include "sprites.h"

#include "config.h"

void Map\_init(struct Map\* self, int w,int h, int playerCount)

{

self->w = w;

self->h = h;

self->map = (struct MapItem\*)malloc(sizeof(struct MapItem)\*w\*h);

int x,y;

for(y=0;y<h;y++)

{

for(x=0;x<w;x++)

{

self->map[y\*w+x].type = 0;

self->map[y\*w+x].burnTime = 0;

self->map[y\*w+x].hasPlayer = 0;

self->map[y\*w+x].hasBomb = 0;

self->map[y\*w+x].blockBombs = 0;

if(rand()%100 < Config\_get()->rockChance)

{

self->map[y\*w+x].type = 2;

}

if((x%2==0 && y%2==0) || x==0 || x==w-1 || y==0 || y==h-1)

self->map[y\*w+x].type = 1;

}

}

self->map[1\*w+1].type = 0;

self->map[1\*w+2].type = 0;

self->map[2\*w+1].type = 0;

self->map[1\*w+w-2].type = 0;

self->map[1\*w+w-3].type = 0;

self->map[2\*w+w-2].type = 0;

self->map[(h-2)\*w+w-2].type = 0;

self->map[(h-2)\*w+w-3].type = 0;

self->map[(h-3)\*w+w-2].type = 0;

self->map[(h-2)\*w+1].type = 0;

self->map[(h-2)\*w+2].type = 0;

self->map[(h-3)\*w+1].type = 0;

for(y=0;y<h;y++)

{

for(x=0;x<w;x++)

{

self->map[y\*w+x].hasBonus = 0;

if(self->map[y\*w+x].type==2)

{

if(rand()%100 > 80)

{

self->map[y\*w+x].hasBonus = rand()%2+1;

}

}

}

}

int i;

for(i=0;i<BOMB\_COUNT;i++)

{

Bomb\_init(&self->bombs[i], self);

self->bombsSem[i] = Semaphore\_init(1);

}

self->mapSem = Semaphore\_init(1);

Bomberman\_init(&self->bombermans[0],1,1,1,0,self,0);

Bomberman\_init(&self->bombermans[1],13,1,1,1,self,(playerCount<=1));

Bomberman\_init(&self->bombermans[2],13,11,1,2,self,(playerCount<=2));

Bomberman\_init(&self->bombermans[3],1,11,1,3,self,(playerCount<=3));

}

void Map\_free(struct Map\* self)

{

free(self->map);

int i;

for(i=0;i<BOMB\_COUNT;i++)

{

Semaphore\_free(self->bombsSem[i]);

}

Semaphore\_free(self->mapSem);

for(i=0;i<4;i++)

{

Bomberman\_free(&self->bombermans[i]);

}

}

void Map\_handleKey(struct Map\* self, int key)

{

int i;

for(i=0;i<4;i++)

{

Bomberman\_handleKey(&self->bombermans[i], key);

}

}

void Map\_draw(struct Map\* self)

{

int x,y;

for(y=0;y<self->h;y++)

{

for(x=0;x<self->w;x++)

{

if(self->map[y\*self->w+x].type==0)

{

if(self->map[y\*self->w+x].burnTime>0)

{

Sprite\_draw(Sprites\_get(S\_FIRE),x\*4,y\*2);

}else

{

if(self->map[y\*self->w+x].hasBonus==0)

{

Sprite\_draw(Sprites\_get(S\_FLOOR),x\*4,y\*2);

}else

if(self->map[y\*self->w+x].hasBonus==1)

{

Sprite\_draw(Sprites\_get(S\_BONUS1),x\*4,y\*2);

}else

if(self->map[y\*self->w+x].hasBonus==2)

{

Sprite\_draw(Sprites\_get(S\_BONUS2),x\*4,y\*2);

}

}

}

if(self->map[y\*self->w+x].type==1)

{

Sprite\_draw(Sprites\_get(S\_WALL),x\*4,y\*2);

}

if(self->map[y\*self->w+x].type==2)

{

Sprite\_draw(Sprites\_get(S\_BRICKS),x\*4,y\*2);

}

}

}

int i;

for(i=0;i<4;i++)

{

Bomberman\_drawGrave(&self->bombermans[i]);

}

for(i=0;i<BOMB\_COUNT;i++)

{

Semaphore\_grab(self->bombsSem[i]);

Bomb\_draw(&self->bombs[i]);

Semaphore\_release(self->bombsSem[i]);

}

for(i=0;i<4;i++)

{

Bomberman\_draw(&self->bombermans[i]);

}

}

void Map\_update(struct Map\* self)

{

int i;

for(i=0;i<BOMB\_COUNT;i++)

{

Semaphore\_grab(self->bombsSem[i]);

Bomb\_update(&self->bombs[i]);

Semaphore\_release(self->bombsSem[i]);

}

int x,y;

for(y=0;y<self->h;y++)

{

for(x=0;x<self->w;x++)

{

self->map[y\*self->w+x].burnTime = self->map[y\*self->w+x].burnTime - 1;

self->map[y\*self->w+x].blockBombs = self->map[y\*self->w+x].blockBombs - 1;

}

}

}

int Map\_hasWall(struct Map\* self, int x, int y)

{

if(x<0 || x>=self->w || y<0 || y>=self->h)

return 1;

int type = self->map[y\*self->w+x].type;

if(type == 0)

return 0;

return 1;

}

int Map\_hasSomething(struct Map\* self, int x, int y)

{

if(Map\_hasWall(self,x,y))

return 1;

int hasPlayer = self->map[y\*self->w+x].hasPlayer;

int hasBomb = self->map[y\*self->w+x].hasBomb;

int hasFire = self->map[y\*self->w+x].burnTime>=0;

if(hasPlayer || hasBomb || hasFire)

return 1;

}

int Map\_wallType(struct Map\* self, int x, int y)

{

if(x<0 || x>=self->w || y<0 || y>=self->h)

return -1;

int type = self->map[y\*self->w+x].type;

return type;

}

int Map\_createBomb(struct Map\* self, int x,int y, struct Bomberman\* b)

{

int i;

int found = -1;

for(i=0;i<BOMB\_COUNT;i++)

{

Semaphore\_grab(self->bombsSem[i]);

if(!self->bombs[i].alive)

{

Bomb\_create(&self->bombs[i],x,y,b);

Semaphore\_release(self->bombsSem[i]);

found = i;

break;

}

Semaphore\_release(self->bombsSem[i]);

}

return found;

}

struct MapItem\* Map\_get(struct Map\* self, int x,int y)

{

if(x<0 || x>=self->w || y<0 || y>=self->h)

return 0;

return &self->map[y\*self->w+x];

}

int Map\_inDanger(struct Map\* self, int x,int y)

{

int i;

for(i=0;i<BOMB\_COUNT;i++)

{

if(self->bombs[i].alive)

{

if(Bomb\_willAffect(&self->bombs[i],x,y))

return 1;

}

}

return 0;

}

int Map\_cantGo(struct Map\* self, int x, int y)

{

if(Map\_hasWall(self,x,y))

return 1;

int hasBomb = self->map[y\*self->w+x].hasBomb;

int hasFire = self->map[y\*self->w+x].burnTime>0;

if(hasBomb || hasFire)

return 1;

}

int Map\_findSafe(struct Map\* self, int x,int y,char\* blackMap, int dist, int returnDir)

{

if(blackMap[y\*self->w+x])

return 99999;

blackMap[y\*self->w+x] = 1;

if(Map\_cantGo(self,x,y) && !returnDir)

{

return 99999;

}

if(!Map\_inDanger(self,x,y))

{

return dist;

}

int m1 = Map\_findSafe(self,x-1,y,blackMap, dist+1, 0);

int m2 = Map\_findSafe(self,x+1,y,blackMap, dist+1, 0);

int m3 = Map\_findSafe(self,x,y-1,blackMap, dist+1, 0);

int m4 = Map\_findSafe(self,x,y+1,blackMap, dist+1, 0);

int min = m1;

if(m2<min)

min = m2;

if(m3<min)

min = m3;

if(m4<min)

min = m4;

if(returnDir)

{

if(min==99999)

return 0;

if(min==m1)

return 1;

if(min==m2)

return 2;

if(min==m3)

return 3;

if(min==m4)

return 4;

return 0;

}else

{

return min;

}

}

int Map\_safeDirection(struct Map\* self, int px,int py)

{

char\* blackMap = malloc(self->w\*self->h\*sizeof(char));

int x,y;

for(y=0;y<self->h;y++)

{

for(x=0;x<self->w;x++)

{

blackMap[y\*self->w+x] = 0;

}

}

int d = Map\_findSafe(self,px,py,blackMap, 0, 1);

free(blackMap);

return d;

}

int Map\_safeDistIfBomb(struct Map\* self, int px,int py, struct Bomberman\* b)

{

char\* blackMap = malloc(self->w\*self->h\*sizeof(char));

int x,y;

for(y=0;y<self->h;y++)

{

for(x=0;x<self->w;x++)

{

blackMap[y\*self->w+x] = 0;

}

}

int id = Map\_createBomb(self,px,py,b);

self->bombs[id].fakebomb = 1;

Map\_get(self,px,py)->hasBomb = 0;

int d = Map\_findSafe(self,px,py,blackMap, 0, 0);

if(id>=0)

{

Bomb\_explode(&self->bombs[id]);

}

free(blackMap);

return d;

}

int Map\_dirToNearestEnemy(struct Map\* self, struct Bomberman\* b)

{

int closest = -1;

int minDist = 9999;

int i;

for(i=0;i<4;i++)

{

if(&self->bombermans[i] != b && self->bombermans[i].alive)

{

int dist = abs(b->x-self->bombermans[i].x) + abs(b->y-self->bombermans[i].y);

if(dist<minDist)

{

minDist = dist;

closest = i;

}

}

}

if(closest==-1 || minDist<=3)

{

return 0;

}

if(abs(b->x-self->bombermans[closest].x) > abs(b->y-self->bombermans[closest].y))

{

if((b->x-self->bombermans[closest].x)>0)

return 1;

else

return 2;

}else

{

if((b->y-self->bombermans[closest].y)>0)

return 3;

else

return 4;

}

}

**Файл «bomberman.h»**

#ifndef H\_BOMBERMAN

#define H\_BOMBERMAN

#include <pthread.h>

struct Map;

struct Bomberman

{

int alive;

int x;

int y;

int id;

int direction;

int shouldBomb;

int showGrave;

int bombCount;

int bombPower;

struct Map\* map;

pthread\_t thread;

/\*ai\*/

int isAi;

int tickAi;

int lastDirection;

};

void Bomberman\_init(struct Bomberman\* self, int x, int y, int alive, int i, struct Map\* map, int isAi);

void Bomberman\_free(struct Bomberman\* self);

void Bomberman\_draw(struct Bomberman\* self);

void Bomberman\_drawGrave(struct Bomberman\* self);

void Bomberman\_handleKey(struct Bomberman\* self, int c);

#endif

**Файл «bomberman.c»**

#include "bomberman.h"

#include "sprites.h"

#include "logger.h"

#include "config.h"

#include <ctype.h>

#include "map.h"

void Bomberman\_aiInit(struct Bomberman\* self)

{

self->lastDirection = -1;

self->tickAi = 0;

}

int opposite\_dir(int dir)

{

if(dir==1)

return 2;

if(dir==2)

return 1;

if(dir==3)

return 4;

if(dir==4)

return 3;

return 0;

}

void Bomberman\_ai(struct Bomberman\* self)

{

int bombChanceBoost = 0;

self->tickAi--;

if(self->tickAi<=0)

{

self->tickAi = 2+rand()%3;

}else

{

return;

}

int safeDir = Map\_safeDirection(self->map, self->x,self->y);

if(safeDir!=0)

{

self->direction = safeDir;

self->lastDirection = self->direction;

self->tickAi = 1;

}else

{

int freeLeft = !Map\_hasSomething(self->map,self->x-1,self->y);

int freeRight = !Map\_hasSomething(self->map,self->x+1,self->y);

int freeUp = !Map\_hasSomething(self->map,self->x,self->y-1);

int freeDown = !Map\_hasSomething(self->map,self->x,self->y+1);

int count = freeLeft+freeRight+freeUp+freeDown;

int lastDirectionFree = 0;

if((self->lastDirection == 1 && freeLeft) || (self->lastDirection == 2 && freeRight) ||

(self->lastDirection == 3 && freeUp) || (self->lastDirection == 4 && freeDown))

{

lastDirectionFree = 1;

}

if(count==1)

bombChanceBoost = bombChanceBoost + 50;

else if(count==2)

bombChanceBoost = bombChanceBoost + 20;

if(count==0)

{

}else

if(count==1)

{

self->direction = freeLeft+freeRight\*2+freeUp\*3+freeDown\*4;

self->lastDirection = self->direction;

}else

if(count == 2 && lastDirectionFree)

{

self->direction = self->lastDirection;

}else

{

int nearestEnemyDirection = Map\_dirToNearestEnemy(self->map,self);

int canGo = ((nearestEnemyDirection==1 && freeLeft) || (nearestEnemyDirection==2 && freeRight)

|| (nearestEnemyDirection==3 && freeUp) || (nearestEnemyDirection==4 && freeDown) );

if(nearestEnemyDirection!=0 && canGo && rand()%100<60)

{

self->direction = nearestEnemyDirection;

self->lastDirection = self->direction;

}else

{

int randDir = 0;

while(1)

{

randDir = rand()%4+1;

if(opposite\_dir(randDir)==self->lastDirection)

continue;

if(randDir==1 && !freeLeft)

continue;

if(randDir==2 && !freeRight)

continue;

if(randDir==3 && !freeUp)

continue;

if(randDir==4 && !freeDown)

continue;

break;

}

self->direction = randDir;

self->lastDirection = self->direction;

}

}

int newx = self->x+(self->direction==2)-(self->direction==1);

int newy = self->y+(self->direction==4)-(self->direction==3);

if(Map\_inDanger(self->map,newx,newy))

self->direction = 0;

}

int newx = self->x+(self->direction==2)-(self->direction==1);

int newy = self->y+(self->direction==4)-(self->direction==3);

int bombChance = rand()%100;

if(bombChance>(90-bombChanceBoost) && self->direction!=0)

{

int dist = Map\_safeDistIfBomb(self->map, self->x,self->y, self);

if(dist<=3)

{

self->shouldBomb = 1;

self->direction = 0;

self->tickAi = 1;

}

}

}

void\* Bomberman\_run(void\* ptr)

{

struct Bomberman\* self = (struct Bomberman\*)ptr;

while(self->alive)

{

if(Map\_get(self->map,self->x,self->y)->burnTime>0)

{

Semaphore\_grab(self->map->mapSem);

self->showGrave = 1;

self->alive = 0;

Map\_get(self->map,self->x,self->y)->hasPlayer = 0;

Semaphore\_release(self->map->mapSem);

break;

}

if(self->isAi)

{

Bomberman\_ai(self);

}

if(self->shouldBomb)

{

self->shouldBomb = 0;

Semaphore\_grab(self->map->mapSem);

if(Map\_get(self->map,self->x,self->y)->hasBomb==0 && self->bombCount>0)

{

self->bombCount--;

Map\_createBomb(self->map,self->x,self->y,self);

}

Semaphore\_release(self->map->mapSem);

}

if(self->direction!=0)

{

int oldx = self->x;

int oldy = self->y;

if(self->direction==3)

{

self->y = self->y - 1;

}

if(self->direction==4)

{

self->y = self->y + 1;

}

if(self->direction==1)

{

self->x = self->x - 1;

}

if(self->direction==2)

{

self->x = self->x + 1;

}

self->direction = 0;

Semaphore\_grab(self->map->mapSem);

int hasPlayer = Map\_get(self->map,self->x,self->y)->hasPlayer;

int hasBomb = Map\_get(self->map,self->x,self->y)->hasBomb;

if(Map\_hasWall(self->map,self->x,self->y) || hasPlayer || hasBomb)

{

self->x = oldx;

self->y = oldy;

}else

{

Map\_get(self->map,oldx,oldy)->hasPlayer = 0;

Map\_get(self->map,self->x,self->y)->hasPlayer = 1;

if(Map\_get(self->map,self->x,self->y)->hasBonus>0)

{

if(Map\_get(self->map,self->x,self->y)->hasBonus==1)

{

self->bombCount++;

}else

{

self->bombPower++;

}

Map\_get(self->map,self->x,self->y)->hasBonus = 0;

}

}

Semaphore\_release(self->map->mapSem);

}

usleep(1000000/10);

}

}

void Bomberman\_init(struct Bomberman\* self, int x, int y, int alive, int i, struct Map\* map, int isAi)

{

self->x = x;

self->y = y;

self->alive = alive;

self->id = i;

self->direction = 0;

self->shouldBomb = 0;

self->map = map;

self->showGrave = 0;

self->bombCount = 2;

self->bombPower = 2;

self->isAi = isAi;

Bomberman\_aiInit(self);

self->thread = 0;

if(self->alive)

{

Map\_get(self->map,self->x,self->y)->hasPlayer = 1;

pthread\_create(&self->thread, NULL, Bomberman\_run, self);

}

}

void Bomberman\_free(struct Bomberman\* self)

{

self->alive = 0;

if(self->thread>0)

{

pthread\_join(self->thread, NULL);

}

}

void Bomberman\_draw(struct Bomberman\* self)

{

if(self->alive)

{

Sprite\_draw(Sprites\_get(S\_BOMBERMAN),self->x\*4,self->y\*2);

}

}

void Bomberman\_drawGrave(struct Bomberman\* self)

{

if(!self->alive && self->showGrave)

{

Sprite\_draw(Sprites\_get(S\_GRAVE),self->x\*4,self->y\*2);

}

}

void Bomberman\_handleKey(struct Bomberman\* self, int d)

{

if(self->isAi)

return;

if(self->alive)

{

int c = d;

if(isalpha(d))

{

c = tolower(c);

}

if(c==Config\_get()->controls[self->id].up)

{

self->direction = 3;

}

if(c==Config\_get()->controls[self->id].down)

{

self->direction = 4;

}

if(c==Config\_get()->controls[self->id].left)

{

self->direction = 1;

}

if(c==Config\_get()->controls[self->id].right)

{

self->direction = 2;

}

if(c==Config\_get()->controls[self->id].bomb)

{

self->shouldBomb = 1;

}

}

}

**Файл «bomb.h»**

#ifndef H\_BOMB

#define H\_BOMB

struct Bomberman;

struct Map;

struct Bomb

{

int alive;

int x,y;

int time;

int power;

int fakebomb;

struct Map\* map;

struct Bomberman\* bomberman;

};

void Bomb\_init(struct Bomb\* self, struct Map\* map);

void Bomb\_create(struct Bomb\* self, int x, int y, struct Bomberman\* b);

void Bomb\_update(struct Bomb\* self);

void Bomb\_draw(struct Bomb\* self);

void Bomb\_explode(struct Bomb\* self);

int Bomb\_willAffect(struct Bomb\* self, int x, int y);

#endif

**Файл «bomb.c»**

#include "bomb.h"

#include "sprites.h"

#include "map.h"

#include "bomberman.h"

#include "config.h"

void Bomb\_init(struct Bomb\* self, struct Map\* map)

{

self->alive = 0;

self->map = map;

}

void Bomb\_create(struct Bomb\* self, int x, int y, struct Bomberman\* b)

{

self->alive = 1;

self->x = x;

self->y = y;

self->time = Config\_get()->bombTimer;

self->bomberman = b;

self->fakebomb = 0;

Map\_get(self->map,self->x,self->y)->hasBomb = 1;

}

void Bomb\_update(struct Bomb\* self)

{

if(self->alive)

{

self->time = self->time - 1;

if(self->time <= 0 || Map\_get(self->map,self->x,self->y)->burnTime>0 || self->fakebomb)

{

Bomb\_explode(self);

}

}

}

void Bomb\_draw(struct Bomb\* self)

{

if(self->alive)

{

if(self->time > (Config\_get()->bombTimer\*2/3))

{

Sprite\_draw(Sprites\_get(S\_BOMB3),self->x\*4,self->y\*2);

}else

if(self->time > (Config\_get()->bombTimer/3))

{

Sprite\_draw(Sprites\_get(S\_BOMB2),self->x\*4,self->y\*2);

}else

{

Sprite\_draw(Sprites\_get(S\_BOMB1),self->x\*4,self->y\*2);

}

}

}

void Bomb\_wave(struct Bomb\* self, int dx, int dy)

{

int power = self->bomberman->bombPower;

int d = 1;

Map\_get(self->map,self->x,self->y)->burnTime = Config\_get()->bombFireCenterTimer;

while(d<=power)

{

int x = self->x+d\*dx;

int y = self->y+d\*dy;

if(Map\_get(self->map,x,y)->blockBombs>0)

{

return;

}else

if(Map\_wallType(self->map,x,y)==0)

{

if(Map\_get(self->map,x,y)->burnTime<Config\_get()->bombFireTimer)

{

Map\_get(self->map,x,y)->burnTime = Config\_get()->bombFireTimer;

}

}else

if(Map\_wallType(self->map,x,y)==2)

{

Map\_get(self->map,x,y)->type = 0;

Map\_get(self->map,x,y)->blockBombs = 5;

return;

}else

{

break;

}

d = d + 1;

}

}

void Bomb\_explode(struct Bomb\* self)

{

if(self->alive && !self->fakebomb)

{

Bomb\_wave(self,1,0);

Bomb\_wave(self,-1,0);

Bomb\_wave(self,0,1);

Bomb\_wave(self,0,-1);

}

self->alive = 0;

Semaphore\_grab(self->map->mapSem);

Map\_get(self->map,self->x,self->y)->hasBomb = 0;

if(!self->fakebomb)

self->bomberman->bombCount++;

Semaphore\_release(self->map->mapSem);

}

int Bomb\_willAffectWave(struct Bomb\* self, int cx, int cy, int dx, int dy)

{

int power = self->bomberman->bombPower;

int d = 1;

while(d<=power)

{

int x = self->x+d\*dx;

int y = self->y+d\*dy;

if(x==cx && y==cy)

return 1;

if(Map\_wallType(self->map,x,y)==0)

{

}else

if(Map\_wallType(self->map,x,y)==2)

{

return 0;

}else

{

break;

}

d = d + 1;

}

return 0;

}

int Bomb\_willAffect(struct Bomb\* self, int x, int y)

{

if(x==self->x && y==self->y)

return 1;

if(self->alive)

{

int a1 = Bomb\_willAffectWave(self,x,y,1,0);

int a2 = Bomb\_willAffectWave(self,x,y,-1,0);

int a3 = Bomb\_willAffectWave(self,x,y,0,1);

int a4 = Bomb\_willAffectWave(self,x,y,0,-1);

return (a1||a2||a3||a4);

}

return 0;

}