SDL flappy bird

Generated by Doxygen 1.9.3

1 SDL_FB 1

1.0.1 Compile . . . 1
1.0.2 Arch install . . . 1
1.0.3 Debian install . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 File Index 5

3.1 File List . . . 5

4 Class Documentation 7

4.1 Bird Struct Reference . . . 7
4.1.1 Detailed Description . . . 7
4.1.2 Member Data Documentation . . . 7
4.1.2.1 Bounds . . . 7
4.1.2.2 gravity . . . 8
4.1.2.3 img . . . 8
4.1.2.4 isJump . . . 8
4.1.2.5 lift . . . 8
4.1.2.6 self_x . . . 8
4.1.2.7 self_y . . . 8
4.1.2.8 time_sinceJump . . . 9
4.1.2.9 velocity . . . 9
4.2 Engine Struct Reference . . . 9
4.2.1 Detailed Description . . . 9
4.2.2 Member Data Documentation . . . 9
4.2.2.1 background_pic . . . 10
4.2.2.2 bird . . . 10
4.2.2.3 current_time . . . 10
4.2.2.4 mFont . . . 10
4.2.2.5 musicEffect . . . 10
4.2.2.6 pipe_index . . . 10
4.2.2.7 pipe_texture . . . 11
4.2.2.8 pipeGen_time . . . 11
4.2.2.9 pipes . . . 11
4.2.2.10 pismo . . . 11
4.2.2.11 renderer . . . 11
4.2.2.12 since_time . . . 11
4.2.2.13 sound . . . 12
4.2.2.14 state . . . 12
4.2.2.15 window . . . 12
4.3 Pipe Struct Reference . . . 12

4.3.1 Detailed Description . . . 12
4.3.2 Member Data Documentation . . . 12
4.3.2.1 botBounds . . . 13
4.3.2.2 botHeight . . . 13
4.3.2.3 free . . . 13
4.3.2.4 isActive . . . 13
4.3.2.5 self_height . . . 13
4.3.2.6 self_width . . . 13
4.3.2.7 self_x . . . 14
4.3.2.8 topBounds . . . 14
4.3.2.9 topHeight . . . 14
4.3.2.10 velocity . . . 14
4.4 Tekst Struct Reference . . . 14
4.4.1 Detailed Description . . . 14
4.4.2 Member Data Documentation . . . 15
4.4.2.1 A . . . 15
4.4.2.2 Bounds . . . 15
4.4.2.3 teksSurface . . . 15
4.4.2.4 tekstColor . . . 15

5 File Documentation **17**
5.1 bird.c . . . 17
5.2 bird.h . . . 17
5.3 engine.c . . . 18
5.4 engine.h . . . 21
5.5 main.c . . . 21
5.6 pipe.c . . . 22
5.7 pipe.h . . . 23
5.8 stary_main.c . . . 23
5.9 text.c . . . 25
5.10 text.h . . . 25

Index **27**

Chapter 1

SDL_FB

1.0.1 Compile

```
make all
./game  /// in order to play user need to press enter and wait few seconds
```

1.0.2 Arch install

```
sudo pacman -S sdl2 sdl2_image sdl2_mixer sdl2_gfx sdl2_net sdl2_ttf
```

1.0.3 Debian install

```
sudo apt install libsdl2-image-dev libsdl2-ttf-dev libsdl2-ttf-2.0.0-0
make cmake autoconf automake
libtool pkg-config libasound2-dev libpulse-dev libaudio-dev libjack-dev
libx11-dev libxext-dev libxrandr-dev libxcursor-dev libxfixes-dev libxi-dev
libxinerama-dev libxxf86vm-dev libxss-dev libgl1-mesa-dev libdbus-1-dev
libudev-dev libgles2-mesa-dev libegl1-mesa-dev libibus-1.0-dev
fcitx-libs-dev libsamplerate0-dev libsndio-dev libwayland-dev
libxkbcommon-dev libdrm-dev libgbm-dev
git clone https://github.com/libsdl-org/SDL
cd SDL
mkdir build
cd build
../configure
make
sudo make install
```

Chapter 2

Class Index

2.1 Class List

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

Bird 7
Engine 9
Pipe 12
Tekst 14

Chapter 3

File Index

3.1 File List

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

bird.c . . . . . ??
bird.h . . . . . ??
engine.c . . . . . ??
engine.h . . . . . ??
main.c . . . . . ??
pipe.c . . . . . ??
pipe.h . . . . . ??
stary_main.c . . . . . ??
text.c . . . . . ??
text.h . . . . . ??

Chapter 4

Class Documentation

4.1 Bird Struct Reference

Public Attributes

- SDL_Texture * img
- SDL_Rect Bounds
- double self_x
- double self_y
- double velocity
- double gravity
- double lift
- double time_sinceJump
- bool isJump

4.1.1 Detailed Description

Definition at line 15 of file bird.h.

4.1.2 Member Data Documentation

4.1.2.1 Bounds

```
SDL_Rect Bird::Bounds
```

Definition at line 17 of file bird.h.

4.1.2.2 gravity

```
double Bird::gravity
```

Definition at line 21 of file bird.h.

4.1.2.3 img

```
SDL_Texture* Bird::img
```

Definition at line 16 of file bird.h.

4.1.2.4 isJump

```
bool Bird::isJump
```

Definition at line 24 of file bird.h.

4.1.2.5 lift

```
double Bird::lift
```

Definition at line 22 of file bird.h.

4.1.2.6 self_x

```
double Bird::self_x
```

Definition at line 18 of file bird.h.

4.1.2.7 self_y

```
double Bird::self_y
```

Definition at line 19 of file bird.h.

4.1.2.8 time_sinceJump

```
double Bird::time_sinceJump
```

Definition at line 23 of file bird.h.

4.1.2.9 velocity

```
double Bird::velocity
```

Definition at line 20 of file bird.h.

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

- bird.h

4.2 Engine Struct Reference

Public Attributes

- struct Tekst * pismo
- SDL_Texture * pipe_texture
- SDL_Texture * background_pic
- SDL_Renderer * renderer
- Mix_Chunk * musicEffect
- Mix_Music * sound
- TTF_Font * mFont
- SDL_Window * window
- GAME_STATE state
- Bird bird
- Pipe pipes [30]
- long long current_time
- long long since_time
- long long pipeGen_time
- int pipe_index

4.2.1 Detailed Description

Definition at line 33 of file engine.h.

4.2.2 Member Data Documentation

4.2.2.1 background_pic

```
SDL_Texture* Engine::background_pic
```

Definition at line 36 of file engine.h.

4.2.2.2 bird

```
Bird Engine::bird
```

Definition at line 43 of file engine.h.

4.2.2.3 current_time

```
long long Engine::current_time
```

Definition at line 45 of file engine.h.

4.2.2.4 mFont

```
TTF_Font* Engine::mFont
```

Definition at line 40 of file engine.h.

4.2.2.5 musicEffect

```
Mix_Chunk* Engine::musicEffect
```

Definition at line 38 of file engine.h.

4.2.2.6 pipe_index

```
int Engine::pipe_index
```

Definition at line 48 of file engine.h.

4.2.2.7 pipe_texture

```
SDL_Texture* Engine::pipe_texture
```

Definition at line 35 of file engine.h.

4.2.2.8 pipeGen_time

```
long long Engine::pipeGen_time
```

Definition at line 47 of file engine.h.

4.2.2.9 pipes

```
Pipe Engine::pipes[30]
```

Definition at line 44 of file engine.h.

4.2.2.10 pismo

```
struct Tekst* Engine::pismo
```

Definition at line 34 of file engine.h.

4.2.2.11 renderer

```
SDL_Renderer* Engine::renderer
```

Definition at line 37 of file engine.h.

4.2.2.12 since_time

```
long long Engine::since_time
```

Definition at line 46 of file engine.h.

4.2.2.13 sound

```
Mix_Music* Engine::sound
```

Definition at line 39 of file engine.h.

4.2.2.14 state

```
GAME_STATE Engine::state
```

Definition at line 42 of file engine.h.

4.2.2.15 window

```
SDL_Window* Engine::window
```

Definition at line 41 of file engine.h.

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

- engine.h

4.3 Pipe Struct Reference

Public Attributes

- int topHeight
- int botHeight
- int free
- int self_width
- int self_height
- double self_x
- double velocity
- SDL_Rect topBounds
- SDL_Rect botBounds
- bool isActive

4.3.1 Detailed Description

Definition at line 14 of file pipe.h.

4.3.2 Member Data Documentation

4.3.2.1 botBounds

```
SDL_Rect Pipe::botBounds
```

Definition at line 23 of file pipe.h.

4.3.2.2 botHeight

```
int Pipe::botHeight
```

Definition at line 16 of file pipe.h.

4.3.2.3 free

```
int Pipe::free
```

Definition at line 17 of file pipe.h.

4.3.2.4 isActive

```
bool Pipe::isActive
```

Definition at line 24 of file pipe.h.

4.3.2.5 self_height

```
int Pipe::self_height
```

Definition at line 19 of file pipe.h.

4.3.2.6 self_width

```
int Pipe::self_width
```

Definition at line 18 of file pipe.h.

4.3.2.7 self_x

```
double Pipe::self_x
```

Definition at line 20 of file pipe.h.

4.3.2.8 topBounds

```
SDL_Rect Pipe::topBounds
```

Definition at line 22 of file pipe.h.

4.3.2.9 topHeight

```
int Pipe::topHeight
```

Definition at line 15 of file pipe.h.

4.3.2.10 velocity

```
double Pipe::velocity
```

Definition at line 21 of file pipe.h.

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

- pipe.h

4.4 Tekst Struct Reference

Public Attributes

- char A [500]
- SDL_Surface * teksSurface
- SDL_Rect Bounds
- SDL_Color tekstColor

4.4.1 Detailed Description

Definition at line 12 of file text.h.

4.4.2 Member Data Documentation

4.4.2.1 A

```
char Tekst::A[500]
```

Definition at line 13 of file text.h.

4.4.2.2 Bounds

```
SDL_Rect Tekst::Bounds
```

Definition at line 15 of file text.h.

4.4.2.3 teksSurface

```
SDL_Surface* Tekst::teksSurface
```

Definition at line 14 of file text.h.

4.4.2.4 tekstColor

```
SDL_Color Tekst::tekstColor
```

Definition at line 16 of file text.h.

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

- text.h

Chapter 5

File Documentation

5.1 bird.c

```
00001 #include "bird.h"
00002 #include <stdbool.h>
00003
00004 void birdConstructor(Bird *x) {
00005     x->Bounds.w = x->Bounds.h = 50;
00006     x->self_x = x->Bounds.x = SCREEN_WIDTH / 3;
00007     x->self_y = x->Bounds.y = SCREEN_HEIGHT / 2;
00008
00009     x->gravity = 15;
00010     x->velocity = 0;
00011     x->lift = -450;
00012     x->time_sinceJump = 0;
00013     x->isJump = false;
00014 }
00015
00016 void birdJump(Bird *x) {
00017     x->isJump = true;
00018 }
00019
00020 bool birdUpdate(Bird *x, double dt) {
00021     x->velocity += x->gravity;
00022     x->time_sinceJump += dt;
00023
00024     if(x->isJump == true && x->time_sinceJump >= 0.25) {
00025         x->velocity = x->lift;
00026         x->time_sinceJump = 0;
00027         x->isJump = false;
00028     }
00029
00030     x->self_y += x->velocity * dt;
00031
00032     if(x->self_y + x->Bounds.h / 2 > SCREEN_HEIGHT) {
00033         x->self_y = SCREEN_HEIGHT - x->Bounds.h / 2;
00034         x->velocity = 0;
00035         return true;
00036     }
00037
00038     if(x->self_y + x->Bounds.h / 2 < 0) {
00039         x->self_y = -x->Bounds.h / 2;
00040         x->velocity = 0;
00041         return true;
00042     }
00043
00044     x->Bounds.x = (int) x->self_x;
00045     x->Bounds.y = (int) x->self_y;
00046     return false;
00047 }
```

5.2 bird.h

```
00001 #ifndef BIRD_H
00002 #define BIRD_H
00003
00004 #include <SDL2/SDL.h>
```

```
00005 #include <SDL2/SDL_events.h>
00006 #include <SDL2/SDL_rect.h>
00007 #include <SDL2/SDL_render.h>
00008 #include <SDL2/SDL_touch.h>
00009 #include <stdbool.h>
00010 #include <stdio.h>
00011
00012 extern const int SCREEN_HEIGHT;
00013 extern const int SCREEN_WIDTH;
00014
00015 typedef struct {
00016     SDL_Texture *img;
00017     SDL_Rect Bounds;
00018     double self_x;
00019     double self_y;
00020     double velocity;
00021     double gravity;
00022     double lift;
00023     double time_sinceJump;
00024     bool isJump;
00025 } Bird;
00026
00027 void birdConstructor(Bird *x);
00028 bool birdUpdate(Bird *x, double dt);
00029 void birdJump(Bird *x);
00030
00031 #endif
```

5.3 engine.c

```
00001 #include "engine.h"
00002 #include "bird.h"
00003 #include "pipe.h"
00004 #include "text.h"
00005
00006 #include <SDL2/SDL.h>
00007 #include <SDL2/SDL_events.h>
00008 #include <SDL2/SDL_image.h>
00009 #include <SDL2/SDL_keycode.h>
00010 #include <SDL2/SDL_rect.h>
00011 #include <SDL2/SDL_render.h>
00012 #include <SDL2/SDL_timer.h>
00013 #include <SDL2/SDL_ttf.h>
00014 #include <SDL2/SDL_video.h>
00015 #include <SDL_mixer.h>
00016 #include <stdbool.h>
00017
00018 // returns true if there is intersection
00019 bool collisionDetection(Bird *b, Pipe *p) {
00020     SDL_Rect *prostokat = malloc(sizeof(SDL_Rect));
00021     return SDL_IntersectRect(&p->topBounds, &b->Bounds, prostokat) || SDL_IntersectRect(&p->botBounds,
      &b->Bounds, prostokat);
00022 }
00023
00024 SDL_Texture* loadTexture(char* path, Engine* e) {
00025     SDL_Texture* texture = NULL;
00026     SDL_Surface* surface = IMG_Load(path);
00027
00028     if(surface == NULL) {
00029         printf("Unable to load image %s! SDL_image Error: %s\n", path,
00030             IMG_GetError());
00031     } else {
00032         texture = SDL_CreateTextureFromSurface(e->renderer, surface);
00033         if (texture == NULL) {
00034             printf("Unable to create texture from surface: %s! SDL_image Error: %s\n", path,
      IMG_GetError());
00035         }
00036         SDL_FreeSurface(surface);
00037     }
00038     return texture;
00039 }
00040
00041 bool initGame(Engine *e) {
00042     SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER | SDL_INIT_AUDIO);
00043     Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 2048);
00044
00045     e->sound = Mix_LoadMUS("sounds/sound.mp3");
00046     e->musicEffect = Mix_LoadWAV("sounds/death_sound.ogg");
00047     e->window = SDL_CreateWindow("Flappy bird", SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
00048     SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_OPENGL);
00049
00050     Mix_PlayMusic(e->sound, -1);
00051
```

```
00052     if(e->window == NULL) {
00053         printf("Window could not be created! SDL_Error: %s\n", SDL_GetError());
00054         return false;
00055     }
00056 
00057     e->renderer = SDL_CreateRenderer(e->window, -1, SDL_RENDERER_ACCELERATED |
      SDL_RENDERER_PRESENTVSYNC);
00058     if(e->renderer == NULL) {
00059             printf("Renderer could not be created! SDL Error: %s\n", SDL_GetError());
00060             return false;
00061     }
00062     if(TTF_Init() == -1) {
00063         printf("TTF_SDL could not initialize: TTF_Error: %s\n", TTF_GetError());
00064         return false;
00065     }
00066 
00067 
00068     e->since_time = 0;
00069     e->current_time = 0;
00070     e->pipe_index = 0;
00071     e->pipeGen_time = 0;
00072     e->state = START_GAME;
00073 
00074     birdConstructor(&e->bird);
00075 
00076     e->mFont = TTF_OpenFont("img/Bullpen3D.ttf", 24);
00077     if(e->mFont == NULL) {
00078         printf("Failed to load font! SDL_ttf Error: %s\n", TTF_GetError());
00079         return false;
00080     }
00081 
00082     SDL_SetRenderDrawColor(e->renderer, 0x00, 0x00, 0x00, 0x00);
00083 
00084     //tekstConstructor(e->pismo, "nacisnij enter aby rozpaczac", e->mFont);
00085     //tekstUpdate(e->pismo, e->mFont, SCREEN_WIDTH/2);
00086     SDL_Delay(100);
00087 
00088     for(int i = 0; i < noPipes; i++) {
00089         pipeConstructor(&e->pipes[i]);
00090     }
00091     return true;
00092 }
00093 
00094 bool loadMedia(Engine *e) {
00095     e->bird.img = loadTexture("img/bird.png", e);
00096     if(e->bird.img == NULL)
00097         return false;
00098     e->pipe_texture = loadTexture("img/pipe.png", e);
00099     if(e->pipe_texture == NULL)
00100         return false;
00101     e->background_pic = loadTexture("img/background.png", e);
00102     if(e->background_pic == NULL)
00103         return false;
00104     return true;
00105 }
00106 
00107 void input(Engine *e, SDL_Event *event) {
00108     while(SDL_PollEvent(event) != 0) {
00109 
00110         if(event->type == SDL_QUIT) {
00111             e->state = QUIT_GAME;
00112         } else if(event->type == SDL_KEYDOWN) {
00113             switch(event->key.keysym.sym) {
00114                 case SDLK_SPACE:
00115                     birdJump(&e->bird);
00116                     break;
00117                 case SDLK_ESCAPE:
00118                     if(e->state == LOST_GAME)
00119                         e->state = QUIT_GAME;
00120                     break;
00121                 case SDLK_RETURN:
00122                     if(e->state == START_GAME)
00123                         e->state = PLAYING;
00124                     if(e->state == LOST_GAME) {
00125                         resetGame(e);
00126                         e->state = PLAYING;
00127                     }
00128                     break;
00129             }
00130         }
00131 
00132         else if(event->type == SDL_KEYUP)
00133             switch (event->key.keysym.sym) {
00134                 case SDLK_SPACE:
00135                     e->bird.isJump = false;
00136                     break;
00137             }
```

```
00138     }
00139 }
00140
00141 void updateGame(Engine *e) {
00142     // calculate delta time
00143     e->since_time = e->current_time;
00144     e->current_time = SDL_GetPerformanceCounter();
00145
00146     double dt = ((e->current_time - e->since_time) * 1000 / (double) SDL_GetPerformanceFrequency()) /
      1000;
00147     int total_time = SDL_GetTicks();
00148
00149     if(e->state == PLAYING) {
00150
00151         if(e->pipe_index < noPipes && total_time > e->pipeGen_time + 2200) {
00152
00153             if(e->pipes[e->pipe_index].isActive == false) {
00154                 e->pipes[e->pipe_index].isActive = true;
00155                 e->pipe_index++;
00156                 e->pipeGen_time = total_time;
00157             }
00158             e->pipe_index %= noPipes;
00159         }
00160
00161         if(birdUpdate(&e->bird, dt)) {
00162             Mix_PlayChannel(-1, e->musicEffect, 0);
00163             e->state = LOST_GAME;
00164         }
00165
00166         for(int i = 0; i < noPipes; i++) {
00167             pipeUpdate(&e->pipes[i], dt);
00168
00169             if(collisionDetection(&e->bird, &e->pipes[i])) {
00170                 e->state = LOST_GAME;
00171                 Mix_PlayChannel(-1, e->musicEffect, 0);
00172             }
00173
00174         }
00175     }
00176 }
00177
00178 void renderFrame(Engine *e) {
00179     SDL_RenderClear(e->renderer);
00180     SDL_RenderCopy(e->renderer, e->background_pic, NULL, NULL);
00181
00182     if(e->state == PLAYING || e->state == LOST_GAME) {
00183         for(int i = 0; i < noPipes; i++) {
00184             if(e->pipes[i].isActive == true) {
00185                 SDL_RenderCopyEx(e->renderer, e->pipe_texture, NULL, &e->pipes[i].topBounds, 0, NULL,
      SDL_FLIP_NONE);
00186                 SDL_RenderCopyEx(e->renderer, e->pipe_texture, NULL, &e->pipes[i].botBounds, 0, NULL,
      SDL_FLIP_VERTICAL);
00187             }
00188         }
00189
00190         SDL_RenderCopy(e->renderer, e->bird.img, NULL, &e->bird.Bounds);
00191     }
00192     SDL_RenderPresent(e->renderer);
00193 }
00194
00195 void closeGame(Engine *e) {
00196     SDL_DestroyTexture(e->bird.img);
00197     e->bird.img = NULL;
00198     SDL_DestroyTexture(e->pipe_texture);
00199     e->pipe_texture = NULL;
00200     SDL_DestroyTexture(e->background_pic);
00201     e->pipe_texture = NULL;
00202     SDL_DestroyRenderer(e->renderer);
00203     e->renderer = NULL;
00204     SDL_DestroyWindow(e->window);
00205
00206     Mix_FreeChunk(e->musicEffect);
00207     Mix_FreeMusic(e->sound);
00208     IMG_Quit();
00209     TTF_Quit();
00210     Mix_CloseAudio();
00211     SDL_Quit();
00212 }
00213
00214 void resetGame(Engine *e) {
00215     birdConstructor(&e->bird);
00216     e->since_time = 0;
00217     e->current_time = SDL_GetPerformanceCounter();
00218     e->pipe_index = 0;
00219     e->pipeGen_time = 0;
00220     for(int i = 0; i < noPipes; i++) {
00221         pipeConstructor(&e->pipes[i]);
```

```
00222     }
00223 }
```

5.4 engine.h

```
00001 #ifndef ENGINE_H
00002 #define ENGINE_H
00003
00004 #include "bird.h"
00005 #include "pipe.h"
00006 #include "text.h"
00007
00008 #include <math.h>
00009 #include <SDL2/SDL_events.h>
00010 #include <SDL2/SDL.h>
00011 #include <SDL2/SDL_image.h>
00012 #include <SDL2/SDL_rect.h>
00013 #include <SDL2/SDL_render.h>
00014 #include <SDL2/SDL_surface.h>
00015 #include <SDL2/SDL_timer.h>
00016 #include <SDL2/SDL_video.h>
00017 #include <SDL_mixer.h>
00018 #include <SDL2/SDL_ttf.h>
00019 #include <stdbool.h>
00020 #include <stdio.h>
00021
00022 extern const int SCREEN_WIDTH;
00023 extern const int SCREEN_HEIGHT;
00024 extern const int noPipes;
00025
00026 typedef enum game_state{
00027     START_GAME,
00028     PLAYING,
00029     LOST_GAME,
00030     QUIT_GAME
00031 } GAME_STATE;
00032
00033 typedef struct {
00034     struct Tekst *pismo;
00035     SDL_Texture *pipe_texture;
00036     SDL_Texture *background_pic;
00037     SDL_Renderer *renderer;
00038     Mix_Chunk *musicEffect;
00039     Mix_Music *sound;
00040     TTF_Font *mFont;
00041     SDL_Window *window;
00042     GAME_STATE state;
00043     Bird bird;
00044     Pipe pipes[30];
00045     long long current_time;
00046     long long since_time;
00047     long long pipeGen_time;
00048     int pipe_index;
00049 } Engine;
00050
00051 bool collisionDetection(Bird *b, Pipe *p);
00052 bool loadMedia(Engine *e);
00053 bool initGame(Engine *e);
00054 void closeGame(Engine *e);
00055 void input(Engine *e, SDL_Event *event);
00056 void updateGame(Engine *e);
00057 void renderFrame(Engine *e);
00058 void resetGame(Engine *e);
00059 SDL_Texture* loadTexture(char *path, Engine *e);
00060
00061 #endif
```

5.5 main.c

```
00001 #include "engine.h"
00002
00003 #include <math.h>
00004 #include <SDL2/SDL_events.h>
00005 #include <SDL2/SDL.h>
00006 #include <SDL2/SDL_image.h>
00007 #include <SDL2/SDL_keycode.h>
00008 #include <SDL2/SDL_rect.h>
00009 #include <SDL2/SDL_render.h>
00010 #include <SDL2/SDL_surface.h>
```

```
00011 #include <SDL2/SDL_timer.h>
00012 #include <SDL2/SDL_video.h>
00013 #include <stdbool.h>
00014 #include <stdio.h>
00015 #include <stdlib.h>
00016
00017 const int SCREEN_WIDTH = 640;
00018 const int SCREEN_HEIGHT = 480;
00019 const int noPipes = 8;
00020
00021 int main(int argc, char **argv) {
00022     srand(time(NULL));
00023     printf("Aby rozpocząć grę wciśnij enter. \n W celu wykonania skoku spacje.  \n");
00024
00025     Engine silnikGry;
00026     if(!initGame(&silnikGry)) {
00027         printf("failed to init!\n");
00028     }
00029     else {
00030         if(!loadMedia(&silnikGry))
00031             printf("failed to init!\n");
00032         else {
00033             SDL_Event imprezka;
00034
00035
00036             while(silnikGry.state != QUIT_GAME) {
00037                 input(&silnikGry, &imprezka);
00038
00039                 updateGame(&silnikGry);
00040
00041                 renderFrame(&silnikGry);
00042             }
00043
00044         }
00045     }
00046     closeGame(&silnikGry);
00047 }
```

5.6 pipe.c

```
00001 #include "pipe.h"
00002 #include <stdlib.h>
00003
00004 void newPipe(Pipe *p) {
00005     p->self_x = SCREEN_WIDTH;
00006     int d = rand()%(SCREEN_HEIGHT-200)+100;
00007
00008     p->botHeight = SCREEN_HEIGHT - (d + p->free / 2);
00009     p->topHeight = d - p->free / 2;
00010
00011     p->botBounds.x = p->self_x;
00012     p->botBounds.y = d + p->free / 2;
00013     p->botBounds.w = p->self_width;
00014     p->botBounds.h = p->self_height;
00015
00016     p->topBounds.x = p->self_x;
00017     p->topBounds.y = d - p->free / 2 - p->self_height;
00018     p->topBounds.w = p->self_width;
00019     p->topBounds.h = p->self_height;
00020 }
00021
00022 void pipeConstructor(Pipe *p) {
00023     p->self_width = 70;
00024     p->self_height = 300;
00025     p->velocity = 120;
00026     p->isActive = false;
00027     p->free = 200;
00028
00029     newPipe(p);
00030 }
00031
00032 void pipeUpdate(Pipe *p, double dt) {
00033     if(p->isActive == false)    return;
00034
00035     p->self_x -= p->velocity * dt;
00036
00037     p->topBounds.x = p->self_x;
00038     p->botBounds.x = p->self_x;
00039
00040     if(p->self_x + p->self_width < 0) {
00041         p->isActive = false;
00042         newPipe(p);
00043     }
00044 }
```

5.7 pipe.h

```
00001 #ifndef PIPE_H
00002 #define PIPE_H
00003
00004 #include "bird.h"
00005 #include <SDL2/SDL.h>
00006 #include <SDL2/SDL_rect.h>
00007 #include <stdbool.h>
00008 #include <stdlib.h>
00009 #include <time.h>
00010
00011 extern const int SCREEN_HEIGHT;
00012 extern const int SCREEN_WIDTH;
00013
00014 typedef struct {
00015     int topHeight;
00016     int botHeight;
00017     int free;
00018     int self_width;
00019     int self_height;
00020     double self_x;
00021     double velocity;
00022     SDL_Rect topBounds;
00023     SDL_Rect botBounds;
00024     bool isActive;
00025 } Pipe;
00026
00027 void pipeConstructor(Pipe *p);
00028 void pipeUpdate(Pipe *p, double dt);
00029
00030 #endif
```

5.8 stary_main.c

```
00001 #include <SDL2/SDL.h>
00002 #include <SDL2/SDL_image.h>
00003 #include <SDL2/SDL_rect.h>
00004 #include <SDL2/SDL_surface.h>
00005 #include <SDL2/SDL_timer.h>
00006 #include <math.h>
00007 #include <stdio.h>
00008
00009 #include "bird.h"
00010
00011 const int WIDTH = 1280;
00012 const int HEIGHT = 720;
00013
00014 int main(int argc, char *argv[]) {
00015   // Declare a pointer
00016   SDL_Window *window;
00017
00018   SDL_Init(SDL_INIT_VIDEO |
00019            SDL_INIT_TIMER); // Initialize SDL2, graphics and timer system
00020
00021   Bird *ptaszek;
00022   birdConstructor(ptaszek);
00023
00024   // structure containg metadata bout background
00025   SDL_Rect background_rect;
00026   background_rect.x = 0;
00027   background_rect.y = 0;
00028   background_rect.w = WIDTH;
00029   background_rect.h = HEIGHT;
00030
00031   // structure containg metadata bout ground
00032   SDL_Rect ground_rect;
00033   ground_rect.x = 0;
00034   ground_rect.y = HEIGHT - 40;
00035   ground_rect.w = WIDTH;
00036   ground_rect.h = 40;
00037
00038   SDL_Rect bird_rect = ptaszek->bounds;
00039
00040   // Create an application window with the following settings:
00041   window = SDL_CreateWindow("Flappy_bird",           // window title
00042                             SDL_WINDOWPOS_CENTERED, // initial x position
00043                             SDL_WINDOWPOS_CENTERED, // initial y position
00044                             WIDTH,                  // width, in pixels
00045                             HEIGHT,                 // height, in pixels
00046                             SDL_WINDOW_OPENGL       // flags - see below
00047   );
00048
```

```
00049   // Check that the window was successfully created
00050   if (window == NULL) {
00051     // In the case that the window could not be made...
00052     printf("Could not create window: %s\n", SDL_GetError());
00053     return 1;
00054   }
00055
00056   // Create a renderer, which sets up the graphics and hardware
00057   Uint32 render_flags = SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC;
00058   SDL_Renderer *rend = SDL_CreateRenderer(window, -1, render_flags);
00059   if (rend == NULL) {
00060     printf("error creating renderer: %s\n", SDL_GetError());
00061     SDL_DestroyWindow(window);
00062     SDL_Quit();
00063     return 1;
00064   }
00065
00067   SDL_Surface *surface_background = IMG_Load("img/background.png");
00068   if (surface_background == NULL) {
00069     printf("error creating surface %s\n", SDL_GetError());
00070     SDL_DestroyRenderer(rend);
00071     SDL_DestroyWindow(window);
00072     SDL_Quit();
00073     return 1;
00074   }
00075
00076   SDL_Surface *surface_ground = IMG_Load("img/ground.png");
00077   if (surface_ground == NULL) {
00078     printf("error creating surface %s\n", SDL_GetError());
00079     SDL_DestroyRenderer(rend);
00080     SDL_DestroyWindow(window);
00081     SDL_Quit();
00082     return 1;
00083   }
00084
00085   SDL_Surface *surface_bird = IMG_Load("img/bird.png");
00086   if (surface_ground == NULL) {
00087     printf("error creating surface %s\n", SDL_GetError());
00088     SDL_DestroyRenderer(rend);
00089     SDL_DestroyWindow(window);
00090     SDL_Quit();
00091     return 1;
00092   }
00093
00094   // Load the image into the hardware's memory
00095   SDL_Texture *tex_background =
00096       SDL_CreateTextureFromSurface(rend, surface_background);
00097   SDL_FreeSurface(surface_background);
00098   if (tex_background == NULL) {
00099     printf("error creating texuture %s\n", SDL_GetError());
00100     SDL_DestroyRenderer(rend);
00101     SDL_DestroyWindow(window);
00102     SDL_Quit();
00103     return 1;
00104   }
00105
00106   SDL_Texture *tex_ground = SDL_CreateTextureFromSurface(rend, surface_ground);
00107   SDL_FreeSurface(surface_ground);
00108   if (tex_ground == NULL) {
00109     printf("error creating texuture %s\n", SDL_GetError());
00110     SDL_DestroyRenderer(rend);
00111     SDL_DestroyWindow(window);
00112     SDL_Quit();
00113     return 1;
00114   }
00115
00116   SDL_Texture *tex_bird = SDL_CreateTextureFromSurface(rend, surface_bird);
00117   SDL_FreeSurface(surface_bird);
00118   if (tex_bird == NULL) {
00119     printf("error creating texuture %s\n", SDL_GetError());
00120     SDL_DestroyRenderer(rend);
00121     SDL_DestroyWindow(window);
00122     SDL_Quit();
00123     return 1;
00124   }
00125
00126   // Clear the windows
00127   SDL_RenderClear(rend);
00128
00129   // Draw the image to the window
00130   SDL_RenderCopy(rend, tex_background, NULL, &background_rect);
00131   SDL_RenderCopy(rend, tex_ground, NULL, &ground_rect);
00132   SDL_RenderCopy(rend, tex_bird, NULL, &bird_rect);
00133   // Double buffering
00134   SDL_RenderPresent(rend);
00135   // The window is open: could enter program loop here (see SDL_PollEvent())
00136
```

```
00137   SDL_Delay(2000); // Pause execution for 3000 milliseconds, for example
00138
00139   // clean up resources before exitng
00140   SDL_DestroyTexture(tex_background);
00141   SDL_DestroyTexture(tex_ground);
00142   SDL_DestroyRenderer(rend);
00143   SDL_DestroyWindow(window);
00144
00145   // Clean up
00146   SDL_Quit();
00147   return 0;
00148 }
```

5.9 text.c

```
00001 #include "text.h"
00002 #include "bird.h"
00003 #include <SDL2/SDL_surface.h>
00004 #include <SDL2/SDL_ttf.h>
00005 #include <string.h>
00006
00007 void tekstUpdate(struct Tekst *S, TTF_Font *f, int pos) {
00008     if(S->teksSurface != NULL) {
00009         SDL_FreeSurface(S->teksSurface);
00010         S->teksSurface = NULL;
00011     }
00012     S->teksSurface = TTF_RenderText_Blended_Wrapped(f, S->A, S->tekstColor, 640);
00013     if(S->teksSurface == NULL) {
00014         printf("Unable to render text surface! SDL_ttf Error: %s\n", TTF_GetError());
00015     } else {
00016         S->Bounds.w = S->teksSurface->w;
00017         S->Bounds.h = S->teksSurface->h;
00018     }
00019
00020     S->Bounds.x = SCREEN_WIDTH / 2 - S->Bounds.w / 2;
00021     S->Bounds.y = pos;
00022 }
00023
00024 bool tekstConstructor(struct Tekst *S, char A[], TTF_Font *f) {
00025     strcpy(S->A, A);
00026
00027     S->tekstColor.r = 255;
00028     S->tekstColor.g = 255;
00029     S->tekstColor.b = 255;
00030     S->tekstColor.a = 255;
00031
00032     S->teksSurface = TTF_RenderText_Blended_Wrapped(f, S->A, S->tekstColor, 640);
00033     if(S->teksSurface == NULL) {
00034                 printf("Unable to render text surface! SDL_ttf Error: %s\n", TTF_GetError());
00035                 return false;
00036     } else {
00037         S->Bounds.w = S->teksSurface->w;
00038         S->Bounds.h = S->teksSurface->h;
00039     }
00040     S->Bounds.x = 0;
00041     S->Bounds.y = 0;
00042     return true;
00043 }
```

5.10 text.h

```
00001 #ifndef TEXT_H
00002 #define TEXT_H
00003
00004 #include <SDL2/SDL.h>
00005 #include <SDL2/SDL_ttf.h>
00006 #include <stdio.h>
00007 #include <stdbool.h>
00008
00009 extern const int SCREEN_WIDTH;
00010 extern const int SCREEN_HEIGHT;
00011
00012 struct Tekst {
00013     char A[500];
00014     SDL_Surface *teksSurface;
00015     SDL_Rect Bounds;
00016     SDL_Color tekstColor;
00017 };
00018
```

```
00019 bool tekstConstructor(struct Tekst *S, char A[], TTF_Font *f);
00020 void tekstUpdate(struct Tekst *S, TTF_Font *f, int pos);
00021
00022 #endif
```

Index

A
    Tekst, 15

background_pic
    Engine, 9
Bird, 7
    Bounds, 7
    gravity, 7
    img, 8
    isJump, 8
    lift, 8
    self_x, 8
    self_y, 8
    time_sinceJump, 8
    velocity, 9
bird
    Engine, 10
botBounds
    Pipe, 12
botHeight
    Pipe, 13
Bounds
    Bird, 7
    Tekst, 15

current_time
    Engine, 10

Engine, 9
    background_pic, 9
    bird, 10
    current_time, 10
    mFont, 10
    musicEffect, 10
    pipe_index, 10
    pipe_texture, 10
    pipeGen_time, 11
    pipes, 11
    pismo, 11
    renderer, 11
    since_time, 11
    sound, 11
    state, 12
    window, 12

free
    Pipe, 13

gravity
    Bird, 7

img
    Bird, 8
isActive
    Pipe, 13
isJump
    Bird, 8

lift
    Bird, 8

mFont
    Engine, 10
musicEffect
    Engine, 10

Pipe, 12
    botBounds, 12
    botHeight, 13
    free, 13
    isActive, 13
    self_height, 13
    self_width, 13
    self_x, 13
    topBounds, 14
    topHeight, 14
    velocity, 14
pipe_index
    Engine, 10
pipe_texture
    Engine, 10
pipeGen_time
    Engine, 11
pipes
    Engine, 11
pismo
    Engine, 11

renderer
    Engine, 11

self_height
    Pipe, 13
self_width
    Pipe, 13
self_x
    Bird, 8
    Pipe, 13
self_y
    Bird, 8
since_time
    Engine, 11

sound
    Engine, 11
state
    Engine, 12

teksSurface
    Tekst, 15
Tekst, 14
    A, 15
    Bounds, 15
    teksSurface, 15
    tekstColor, 15
tekstColor
    Tekst, 15
time_sinceJump
    Bird, 8
topBounds
    Pipe, 14
topHeight
    Pipe, 14

velocity
    Bird, 9
    Pipe, 14

window
    Engine, 12