Ikaruga

INF-CPP praktikum1

Inhalt

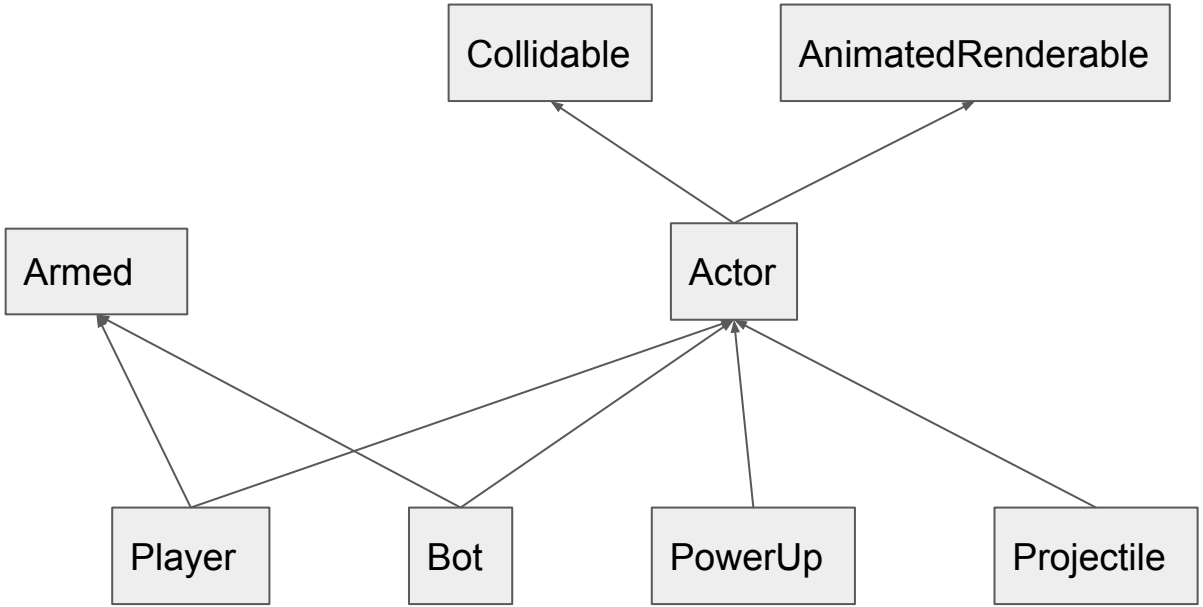
- Game Design
- XML
- Editor
- Menü/UI
- Collision
- Bots
- Weapons
- PowerUps
- Sonstiges
- Demo

Game Design

Spielprinzip

- Sidescroller
- Farben
- PowerUps (instant, duration)
  - Weapons
- Boss (scroll stop)
- Highscore

Struktur


Collidable
AnimatedRenderable
Armed
Actor
Player
Bot
PowerUp
Projectile

XML

von Sven, Patrick S.

- als Bibliothek ausgelagert
- Boost als Grundlage
  - Beispielcode aus Main.cpp
- Datenkern des Spiels
  - Levels, Bots,

    Waffen, PowerUps,

    Profile

```
void XML::load()
{
    std::size_t found = XML::getFilename().find_last_of("/\\");
    string path = XML::getFilename().substr(0, found);

    ptree pt;
    try
    {
        read_xml(XML::getFilename(), pt);
    }
    catch (boost::exception_detail::clone_impl<boost::exception_detail::error_info
    {
        std::cerr << boost::diagnostic_information(e);
        throw std::invalid_argument("Invalid path or filename.");
    }

    try
    {
        BOOST_FOREACH(const ptree::value_type& v, pt.get_child("level"))
                    {
                        if (v.first == "id")
                        {
                            m_id = v.second.get<int>("");
                            m_requiredAttributes["id"]++;
                        }
                        else if (v.first == "name")
```

```
void XML::save()
{
    ptree root;
    ptree level;
    ptree tileset;
    ptree explosions;
    ptree background;
    ptree player;
    ptree statusbar;

    /* Adding Level Information */
    level.put("id", m_id);
    level.put("name", m_levelname);

    /* Adding Tileset */
    tileset.put("<xmlattr>.filename", m_tileset);

    level.add_child("tileset", tileset);

    /* Adding Explosions */
    explosions.put("<xmlattr>.filename", m_explosions);

    level.add_child("explosions", explosions);
```

```
struct Weapon
{
    std::string type;
    std::string filename;
    int colorOffsetX;
    int colorOffsetY;
    int shootingVolume;
    int soundVolume;
    int frameWidth;
    int frameHeight;
    float weaponOffsetX;
    float weaponOffsetY;
    float cooldown;
    std::string soundfile;
    int collisionDamage;
    float speed;
    int numFrames;
};

struct NPC
```

Editor

Wer: Johann Arndt, Patrick Nolte

- XML mit XML-lib geladen und gespeichert.
- LevelScene beim öffnen zur Datenrepräsentation und für die Darstellung
- LevelScene erbt von QGraphicScene
- Texturen erben von QGraphicsItem und enthalten Typinformationen.
- Bei klick auf Textur/Item/Bot zwischenspeichern der Information
- Bei Klick/Drag wird zwischengespeichertes auf die QGraphicScene gesetzt

Editor

- Items/Bots in Vector zwischengespeichert
- TexturIDs in Vectorarray
- Bei Mausmove-event prüfen, ob anderes Tile. Wenn ja dort ebenfalls Textur/Item/Bot setzen/löschen.
- Hintergrund erst gerendert, Objekt danach
- Genaueren Infos von Bots/Items als Tooltips
- Button zum Größe ändern

Menü/UI

Wer: Patrick S., Marius, Patrick N., Jochen, Timo

Wie umgesetzt:

UI: Statusbar, Healthbar, Highscore, Waffen-Stufe

Menü: Filesystem, MainMenu, RenderTable, HighScore

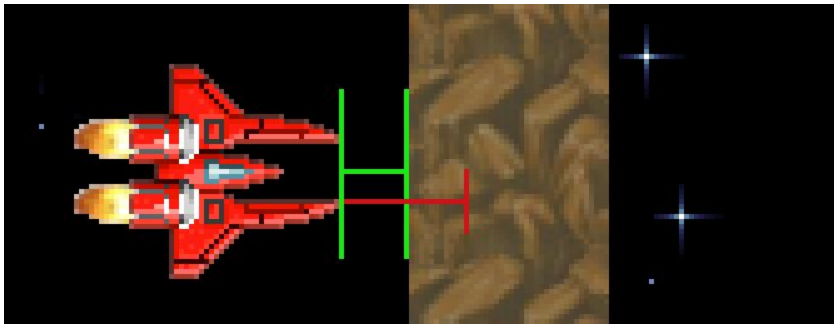
Collision

- Wer? David, Benjamin und Nathan
- **Unterteilung**:
  - Tile-Collision (David & Nathan)
  - Actor-Collision (Benjamin)
- Gemeinsames **Interface** zur Benachrichtigung

```
/**
 * Constructor for Collidable inits the vars
 */
Collidable();

/**
 * Destructor of Collidable
 */
virtual ~Collidable();

/**
 * Is invoked if the actor (Collidable) collides with an tile
 */
virtual void onTileCollision() = 0;

/**
 * Is invoked if the actor collides with another actor
 * It is pure virtual, since the subclasses react differently on
 * collisions with different actors.
 *
 * @parameter other The actor instance which collided with this instance
 */
virtual void onActorCollision(Actor& other) = 0;
```

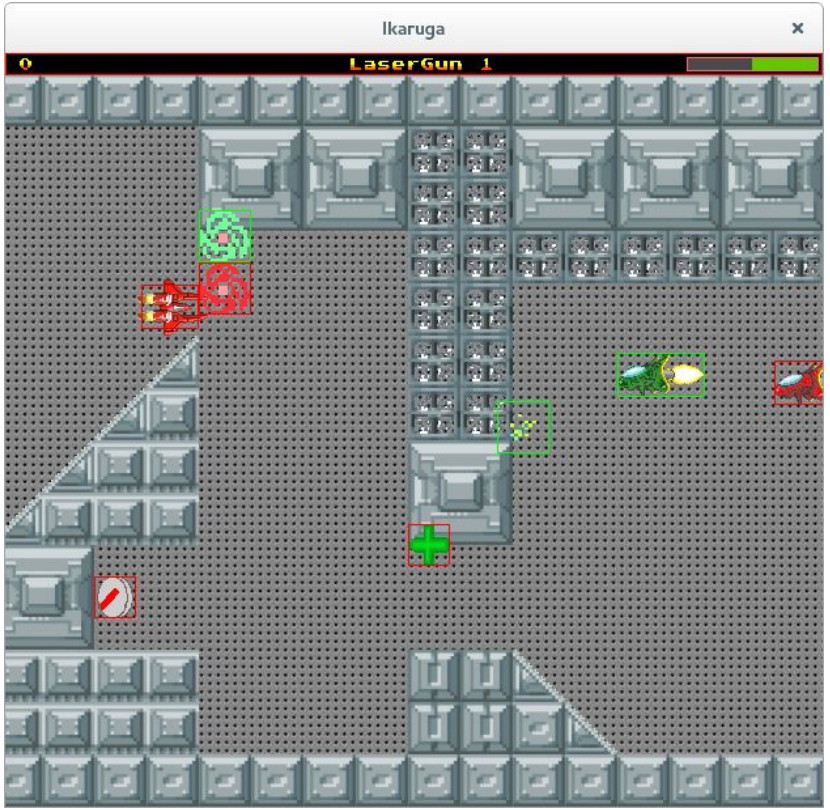
Kollision zwischen Actors und Level

- Wer? David und Nathan
- **Look-ahead** anstatt Look-behind (vorausschauend)
- Erst x-, dann y-Bewegung behandeln
- Umgebende und eigene Tiles überprüfen
- **Fallunterscheidung**, z.B. Kollision mit festem Block
- Differenz zum Block berechnen
- Bewegung auf Differenz einschränken
- Bei Kollision, Aufruf von **Actor::onTileCollision()**



Actor Collision

- Hitboxes
- Zwei Phasen
  - Broad Phase
  - Narrow Phase
- Sweep-and-Prune Algorithmus
  - Vorsortierung von Actors
- Bei Kollision, Aufruf von
  - Actor::onActorCollision(Actor& other)
  - … bei beiden Actors
- Actor reagieren autonom auf Kollision

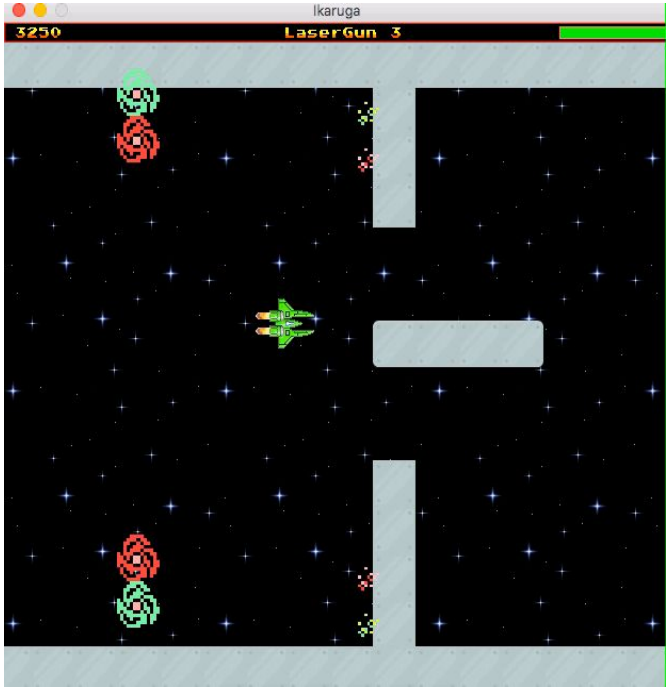


Bots

Wer: Timo, Jochen, Thorsten, Dennis

- Erben von Actor (AnimatedRenderable, Collidable) und Armed
- Unterscheidung
  - Normal Bots
  - Boss Bots (z.B. Endboss)

- Bewegungsmuster: enum BotMoveType{SIN, AI, CIRCLE}
- Erst spawn wenn nahe von der Kamera
- Funktion wie lange Bot aktiv, um f(t) für die Bewegung zu berechnen
- Intelligentes Schussmuster



Weapons

Wer: Johan, Dennis, Jochen, Thorsten

Klassen

```
class Bot : public Actor, public Armed
```

- Armed
  - Mehrfachvererbung
  - Stellt Schießfunktionalität bereit
- Weapon
  - Repräsentiert Waffe mit Infos über Projektile
- Projectile
  - Leitet von Actor ab
  - Hat Richtung, Schaden, Geschwindigkeit, usw.
  - Zieht Hitbox hinter sich her (Bullet Through Paper Problem)

Weapons

Nutzung

- Definition: weapons.xml
- Instantiiert Subklasse von Weapon
- Verwendung über Type (z. B. LASER_GUN)
- Werden dynamisch an Armed gebunden

```
<npc type="NORMAL">
    <move function="AI">30</move>
    <speed>0</speed>
    <stdWeapon>LASER_GUN</stdWeapon>
</npc>
```

PowerUps (PU)

- Bei Kollision übernimmt Player Kontrolle über das PU
  - Static Cast von Actor zu PU
  - Speichern des Objekts in Player-eigenen Vector
  - Entfernen des PU aus der Actor-Liste vom Game
- Bei jedem Update, Aufruf von PowerUp::consume(Player& p)
  - PU manipuliert die Eigenschaften des Players
- Besitzen Verfallsdatum
  - Bei Erreichen werden PUs aus dem Player-eigenen Vector entfernt
  - Ressourcen werden freigegeben

```
void Player::onActorCollision(Actor& other)
{
    if (other.type() == POWERUP)
    {
        PowerUp* powerUp = static_cast<PowerUp*>(&other);
        m_powerUps.push_back(powerUp);
        m_game.removeActor(powerUp);
    }
}
```

PowerUps - Beispiel

```cpp
void Player::consumePowerUps()
{
    vector<PowerUp*> to_remove;

    for (auto powerUp : m_powerUps)
    {
        // consume collected powerup at least once
        powerUp->consume(*this);

        // check if powerup has expired
        if (getLiveTime() > powerUp->getExpirationTime())
        {
            powerUp->stop(*this);
            to_remove.push_back(powerUp);
        }
    }
```

```cpp
void PowerUpGodMode::consume(Player& player)
{
    player.setGodMode(true);
}

void PowerUpGodMode::stop(Player& player)
{
    player.setGodMode(false);
}
```

Sonstiges

- Sound
  - Jochen, Sven
- Grafik
  - Dennis, Thorsten, Jochen
- Animationen
  - Benjamin, Johan
- Dynamisches Einlesen und Erstellen von Aktoren
  - Marius, Timo
- Orga, Hilfe, Pull Requests
  - Benjamin, Jochen, Johan