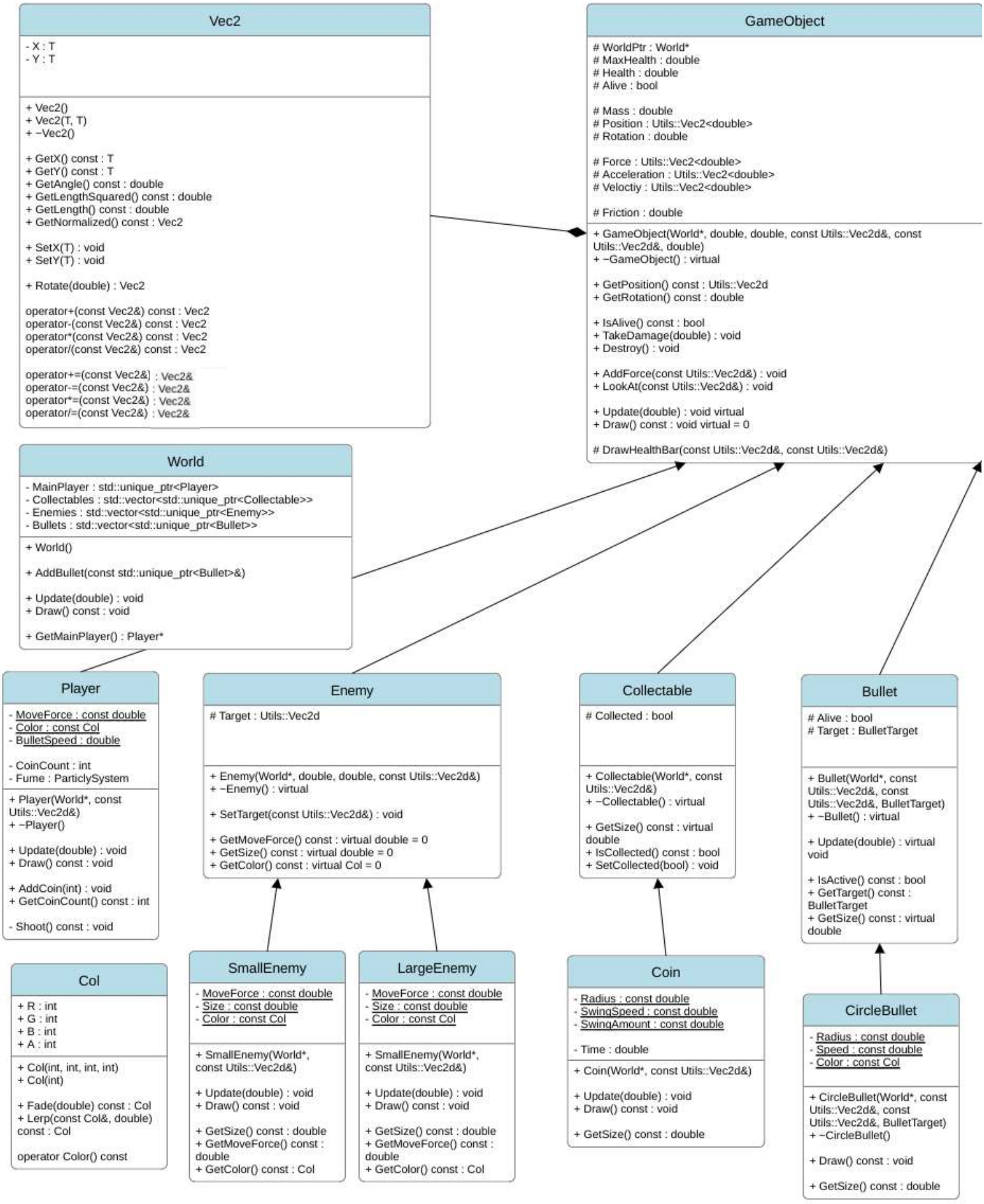
Vec2
- X : T
- Y : T
+ Vec2()
+ Vec2(T, T)
+ ~Vec2()
+ GetX() const : T
+ GetY() const : T
+ GetAngle() const : double
+ GetLengthSquared() const : double
+ GetLength() const : double
+ GetNormalized() const : Vec2
+ SetX(T) : void
+ SetY(T) : void
+ Rotate(double) : Vec2
operator+(const Vec2&) const : Vec2
operator-(const Vec2&) const : Vec2
operator*(const Vec2&) const : Vec2
operator/(const Vec2&) const : Vec2
operator+=(const Vec2&) : Vec2&
operator-=(const Vec2&) : Vec2&
operator*=(const Vec2&) : Vec2&
operator/=(const Vec2&) : Vec2&
GameObject
WorldPtr : World*
MaxHealth : double
Health : double
Alive : bool
Mass : double
Position : Utils::Vec2<double>
Rotation : double
Force : Utils::Vec2<double>
Acceleration : Utils::Vec2<double>
Velocity : Utils::Vec2<double>
Friction : double
+ GameObject(World*, double, double, const Utils::Vec2d&, const Utils::Vec2d&, double)
+ ~GameObject() : virtual
+ GetPosition() const : Utils::Vec2d
+ GetRotation() const : double
+ IsAlive() const : bool
+ TakeDamage(double) : void
+ Destroy() : void
+ AddForce(const Utils::Vec2d&) : void
+ LookAt(const Utils::Vec2d&) : void
+ Update(double) : void virtual
+ Draw() const : void virtual = 0
# DrawHealthBar(const Utils::Vec2d&, const Utils::Vec2d&)
World
- MainPlayer : std::unique_ptr<Player>
- Collectables : std::vector<std::unique_ptr<Collectable>>
- Enemies : std::vector<std::unique_ptr<Enemy>>
- Bullets : std::vector<std::unique_ptr<Bullet>>
+ World()
+ AddBullet(const std::unique_ptr<Bullet>&)
+ Update(double) : void
+ Draw() const : void
+ GetMainPlayer() : Player*
Player
- MoveForce : const double
- Color : const Col
- BulletSpeed : double
- CoinCount : int
- Fume : ParticlySystem
+ Player(World*, const Utils::Vec2d&)
+ ~Player()
+ Update(double) : void
+ Draw() const : void
+ AddCoin(int) : void
+ GetCoinCount() const : int
- Shoot() const : void
Enemy
Target : Utils::Vec2d
+ Enemy(World*, double, double, const Utils::Vec2d&)
+ ~Enemy() : virtual
+ SetTarget(const Utils::Vec2d&) : void
+ GetMoveForce() const : virtual double = 0
+ GetSize() const : virtual double = 0
+ GetColor() const : virtual Col = 0
Collectable
Collected : bool
+ Collectable(World*, const Utils::Vec2d&)
+ ~Collectable() : virtual
+ GetSize() const : virtual double
+ IsCollected() const : bool
+ SetCollected(bool) : void
Bullet
Alive : bool
Target : BulletTarget
+ Bullet(World*, const Utils::Vec2d&, const Utils::Vec2d&, BulletTarget)
+ ~Bullet() : virtual
+ Update(double) : virtual void
+ IsActive() const : bool
+ GetTarget() const : BulletTarget
+ GetSize() const : virtual double
Col
+ R : int
+ G : int
+ B : int
+ A : int
+ Col(int, int, int, int)
+ Col(int)
+ Fade(double) const : Col
+ Lerp(const Col&, double) const : Col
operator Color() const
SmallEnemy
- MoveForce : const double
- Size : const double
- Color : const Col
+ SmallEnemy(World*, const Utils::Vec2d&)
+ Update(double) : void
+ Draw() const : void
+ GetSize() const : double
+ GetMoveForce() const : double
+ GetColor() const : Col
LargeEnemy
- MoveForce : const double
- Size : const double
- Color : const Col
+ SmallEnemy(World*, const Utils::Vec2d&)
+ Update(double) : void
+ Draw() const : void
+ GetSize() const : double
+ GetMoveForce() const : double
+ GetColor() const : Col
Coin
- Radius : const double
- SwingSpeed : const double
- SwingAmount : const double
- Time : double
+ Coin(World*, const Utils::Vec2d&)
+ Update(double) : void
+ Draw() const : void
+ GetSize() const : double
CircleBullet
- Radius : const double
- Speed : const double
- Color : const Col
+ CircleBullet(World*, const Utils::Vec2d&, const Utils::Vec2d&, BulletTarget)
+ ~CircleBullet()
+ Draw() const : void
+ GetSize() const : double
ParticleSystem
- Particle : std::vector<Particle>
+ ParticleSystem()
+ ~ParticleSystem()
+ Emit(const Utils::Vec2d&, const Utils::Vec2d&, double, double, const Col&, const Col&, double, double, int)
+ Update(double) : void
+ Draw() const : void
Particle
+ Position : Utils::Vec2d
+ Velocity : Utils::Vec2d
+ Age : double
+ LifeTime : double
+ StartColor : Col
+ EndColor : Col
+ StartSize : double
+ EndSize : double
+ Particle(const Utils::Vec2d&, const Utils::Vec2d&, double, const Col&, const Col&, double, double)
+ Update(double) : void
+ Draw() const : void

A program felépítése a következő:

-A main file-ban megnyitok egy ablakot Raylibbel, utána létrehozok egy World-öt és amíg nyitva van az ablak, meghívom az Update és Draw metódusait.

-A világ tartalmazza a játékost, az összes Coin-t, Enemy-t és Bullet-ot.

-A játékos figyeli a billentyűzetet és az egeret a mozgáshoz, az Enemy-k pedig automatikusan mozognak a játékos felé. A Coin-ok és a Bullet-ok szintén maguk mozognak.

-Minden objektum, többek között azok is, amiket felsoroltam a GameObject osztályból származik. A GameObject osztály pedig tartalmaz egy World*-t, ami az őt tartalmazó osztályra mutat. Ez azért szükséges, mert ha a játékos lő, akkor létrehoz egy Bullet-ot, és hozzá kell adnia a World-höz, amiben van. Lehet, hogy lenne ennél jobb megoldás is, de azért nincs vészesen sok objektum, szóval egy pointer talán nem nagy gond (viszont nagyon megkönnyíti a dolgom).

-A World.Update update-el mindent (mozgat), ezen felül ellenőrzi a következő ütközéseket:
Player - Collectable, Player - Bullet (ilyen egyébként nincs végül), Enemy - Bullet, Player - Enemy
Ha ütközés történt, törli a megfelelő objektumokat.

-A ParticleSystem egy helyen jelenik meg, méghozzá a játékos hoz létre egy példányt, és amikor mozog, meghívja az Emit metódust, ami kibocsát néhány Particle-t. Természetesen a játékos Update metódusa meghívja a ParticleSystem Update metódusát, az pedig a Particle-ök Update metódusait, így egy idő után a Particle-ök eltűnnek (a ParticleSystem törli a öreg Particle-öket)

-A különböző méretű ellenfelek valóban csak a méretükben és a sebzésükben térnek el (de egy kicsit máshogy is vannak rajzolva), így tényleg valamennyire felesleges az öröklődés, de eredetileg terveztem ennél nagyobb eltérést is

-Nagyjából mindenhez írtam egy rövid tesztet, amelyek GTest-et használnak. A Raylib függőséget sajnos nem nagyon lehet kiküszöbölni, mert a program számos része - még a nem grafikus részek is - használnak Raylib-es függvényeket, és építenek arra, hogy az ablak inicializálva van. Például amikor új ellenfeleket hoz létre a világ, akkor lekérdezi az ablak méretét. (Ha nem lenne inicializálva az ablak, ezt nem tudná megtenni, de használok például ütközés ellenörző függvényeket, illetve Raylib struktúrákat is használok stb.)

-Memóriaszivárgás nincs, ezt az AddressSanitizer segítségével ellenőríztem.

-A CMakeLists.txt-t végül teljesen újraírtam, egy libet hozok létre a kódból, majd ezt linkelem magához a játékhoz, és egy teszt programhoz is. Az assets is átkerült a gyökérbe.

-Ajánlom a shell script-ek használatát (run_game.sh, run_test.sh), ezek lefordítják és le is futtatják az adott programot.

-A program forráskódja fönt van GitHubon: https://github.com/mandliors/Prog2_NHF