Entwicklerdokumentation

Benötigte Headerdateien:

- SDL.h: Ermöglicht die Verwendung von Grafik, Audio, Rendering, Fenster
- SDL_image.h: Ermöglicht das Laden von Grafiken
- SDL_mixer.h: Ermöglicht das Spielen von Musik und Sounds
- SDL_ttf.h: Ermöglicht das Abbilden von Text am Spielfenster

Erstellung eigener Headerdateien:

1. Vector.h: Integration von Vektoren, deren Eigenschaften und die Formeln wie Skalarprodukt, Einheitsvektorenerzeugung, 2er Norm, Berechnung des Winkels zwischen Vektoren; Sinn: Mögliche Zuweisung von Positionen, Punkte und Richtungen von zukünftigen Strukturen
2. Transform.h: Integration von Matrizen, bestehend aus 2 Vektoren, die die Position und die Richtung des „Besitzers“ dieser Matrix angeben. Sinn: Extreme Relevanz der Bewegungsfähigkeit von Strukturen
3. Collider.h: Integration von Kreisen und Rechtecken mit der Fähigkeit, zu überprüfen, ob sie überlappen. Sinn: Prüfungsmöglichkeit von Kollisionen
4. Draw.h: Integration des Ladens und des Zeichnens von Texturen. Sinn: Zeichnen aller graphischen Objekte im Spiel
5. Bullet.h: Einbau von Kugeln mit den Eigenschaften (außer ihrem Transform, Textur und Kreiscolliders): Schaden, Geschwindigkeit, Reichweite, Rückstoß. Sinn: Das Spiel ist ein Top-Down-Shooter
6. Weapon.h: Einbau von Waffen mit den Eigenschaften (außer ihrem Transform und Textur (keine Kollisionen)): Schaden der Kugel, Menge der Munition, Reichweite der Kugel, Rückstoß der Kugel, Cooldown nach Schuss, Schussfähigkeit. Sinn: Die Waffe ist ein wichtiger Bestandteil des Spiels
7. Player.h: Einbau des Spielers mit der Fähigkeit (außer ihrem Transform, Textur und Kreiscolliders): Sich zu bewegen, eine Waffe mit sich zu tragen, Pickups aufzusammeln
8. Enemy.h: Einbau von Zombies mit der Fähigkeit: Sich direkt zum Spieler zu bewegen, von Waffen getroffen werden, getötet werden und Pickups und Blut droppen zu können. Eigenschaften (außer ihrem Transform, Textur und Kreiscolliders): Leben, Bewegungsgeschwindigkeit, Rückstoßfähigkeit.
9. Blood.h: Einbau von Blut, welches sich nicht bewegt. Sinn: Übertriebendes Design des Spiels wird deutlich und zeigt, wie lang der Spieler schon überlebt hat
10. Pillar.h: Einbau von Säulen (manche meinen, es seien Fässer) mit der Fähigkeit: Spieler zurückzuschieben, Zombies zurückzuschieben und Kugeln bei Kollision zu zerstören. Sinn: Mehr Dynamik im Spiel anstatt „Zombies jagen Spieler; er verteidigt sich“
11. Map.h: Zusammenfassung aller existierenden Säulen in einer Struktur. Sinn: Somit ist es leichter, dass jede Säule im Spiel prüfen kann, wer momentan die Säule berührt, um dadurch Gegenmaßnahmen ergreifen zu können

12. Pickup.h: Einbau von Items, die für bestimmte Sorten von Waffen stehen. Eigenschaften (außer ihrem Transform, Textur und Kreiscolliders): Die Klasse der Waffe, die als enum integriert ist.
13. Lists.h: Integration von Pseudolisten, die jeweils Zombies, Kugeln und Pickups enthalten. Es sind eigentlich Arrays aus Pointern der Größe 1000, deren Zugriff allerdings nur auf die Anzahl der Pointern beschränkt ist, die nicht NULL sind. Die Fähigkeiten sind: Erstellen, hinzufügen, entfernen, auf Kollsionen von jedem Objekt prüfen und jedes enthaltende Objekt zu rendern. Sinn: Da während der Laufzeit der Zugriff auf jedem Zombie, Kugel und Pickup benötigt ist, um auf Kollisionen dieser Variablen zu prüfen, müssen diese in Listen eingespeichert werden

Durchführung des main():

Die Funktion besteht aus 3 Funktionen: startGame(), update(), welches nach startGame in einer Schleife mit einer Verzögerung durchgeführt wird, und loadGameOver(), wenn die Abbruchbedingung der Schleife wahr ist.

startGame(): Lädt alle globale Variablen wie der Spieler, die Musikdateien, die Listen, das Fenster, der Renderer, der Schriftart, … . Zudem werden wichtige Funktionen vor dem eigentlichen Spiel durchgeführt wie das Laden des Fensters und das Aktivieren der Musikchannels.

Update(): Besteht aus 4 Funktionen: checkInputs(), applyUpdates(), checkColliders() und renderObjects():

checkInputs(): Prüft, welches Tasten der Spieler gedrückt hat und speichert sie dem entsprechend.

applyUpdates(): Alle physikalischen Operationen wie die Bewegungen aller Objekte im Spiel und das Umsetzen der gespeicherten Tasten werden durchgeführt.

checkColliders(): Prüft alle relevanten Kollisionen ab und setzt dem entsprechend die korrekte Reaktion hinzu:

Kollisionstest (Spieler, Zombie – Säule): Die Lebewesen werden von der Säule weggedrückt.

Kollisionstest (Kugel – Zombie): Die Kugel überträgt den Schaden an den Zombie und dass verbleibende Leben wird korrigiert. Wenn der Zombie stirbt, wird an der Todesposition Blut erzeugt und die Chance, dass eine Waffe liegt.

Kollisionstest (Spieler - Pickup): Die Eigenschaften des enthaltenden Pickups werden an die Waffe übertragen.

Kollisionstest (Spieler - Zombies): Der Spieler stirbt und das Game Over wird geladen.

Da Kugeln und Zombies im Spiel unterschiedlich oft existieren, werden diese Listen verwendet, um jedes Objekt abzudecken.

renderObjects: Bildet jede Grafik von jedem Objekt in das Fenster ab: Spieler, Säulen, Zombies, Kugeln, Blut und den Text, das die Anzahl der Munition angibt.

loadGameOver(): Führt für 2 Sekunden lang das Update() durch, ohne auf Tasten zu überprüfen, d.h., der Spieler bewegt sich nicht mehr; wird das Endergebnis wird in der Konsole dargestellt (wie viele Zombies es getötet haben und es wie viele Zombies getötet hat).