|  |
| --- |
| Kerem Okumus, Moritz Biedenbacher, Kiriakos Avramidis |
| [Las Vegas] |
| Programmbeschreibung über die Software |

|  |
| --- |
| Kerem Okumus, Moritz Biedenbacher, Kiriakos Avramidis  [Datum] |

# Vorwort

Wir haben uns entschieden, dieses Projekt grafisch darzustellen, damit es für Sie einfacher ist, das Spiel zu spielen und es auch für uns unkomplizierter ist, grafisch etwas darzustellen, anstatt es auf der Konsole es zu tun.

Die dazu benutzte Bibliothek heißt SFML (Simple and Fast Multimedia Library) und wird im ganzen Projekt für alles, was Sie sehen werden, benutzt. Die Bibliothek ermöglicht uns, Inhalte grafisch und akustisch darzustellen und das mit ein paar Zeilen Code. Jedoch konnte Ich aufgrund des Spiellogiks nicht sonderlich viel Code ersparen und das ganze Projekt hat einen Zeilenumfang von ca. 9000 Zeilen Code. Jedoch habe ich probiert den kompletten Quelltext so einfach verständlich wie möglich zu halten und haben jede Gruppe an Code, die eine Verknüpfung haben, kommentiert. Trotzdem hatten wir bei der Programmierung sehr viel Spaß, auch wenn wir manchmal an Hindernisse stehen geblieben sind (wir saßen allein an den Würfeln 6,5 Stunden, damit es zufällig gezogen wird) und haben vieles auf dieser Reise gelernt.

Da dieses Projekt sehr komplex scheinen kann, haben wir Ihnen einen kleinen Hilfskasten bereitgestellt, der Ihnen zur Hilfe dienen soll, falls Sie mit dem Code nicht zurechtkommen sollten. Aber Sie sollten es mit dieser Dokumentation den Code sehr einfach verstehen können.

## Einführung in SFML

Simple and Fast Multimedia Library (SFML) ist ein objektorientiertes Open-Source-Multimedia-Framework. Es ist in C++ geschrieben und greift intern auf betriebssystemspezifische Funktionen sowie externe Bibliotheken zurück.

Es wurde mit der Intention entwickelt, möglichst benutzerfreundliche und effiziente Multimedia-Programmierung auf hohem Level zu erlauben, daher auch der Name der Bibliothek.

SFML ist in mehrere Teilpakete gegliedert, die unterschiedliche Aufgabenbereiche erfüllen. Zu den wichtigen Merkmalen gehören:

Es eignet sich hauptsächlich für Spiele und nutzt für die Grafik OpenGL, welches man auch für die Entwicklung von 3D-Anwendungen benutzen könnte. Je nach Anforderungen können die gewünschten Teilpakete einzeln angebunden werden, so reicht zum Beispiel allein das System-Package für parallele Programmierung.

# Nutzung SFML mit Visual Studio 2019

Um Ihnen die Benutzung des Projekts mit SFML zu vereinfachen, ist hier eine kurze Anleitung dazu, wie man SFML mit Visual Studio verknüpft.

**Voraussetzungen**: Ein System mit 64-Bit Betriebssystem und Visual Studio 2017/2019

1. Als Erstes müssen Sie sich SFML herunterladen. Dies geschieht auf der Seite <https://www.sfml-dev.org/download/sfml/2.5.1> und von da aus, laden Sie sich bitte die Visual Studio C++ 15 (2017) 64-Bit herunter und entpacken Sie bitte den Zip-Ordner dahin, wohin Sie möchten (Empfohlen: Desktop oder Dokumente).
2. Laden Sie sich den vollständigen Projektordner von uns herunter. Wir haben unser Projekt auf GitHub hochgeladen, um alles im Blick und für jeden verfügbar zu halten. Die Seite lautet <https://github.com/Turksat46/Las-Vegas>, öffnen Sie die Seite und drücken Sie dann den grünen „Code“-Knopf und dann auf „Download Zip“. Sehr gerne können Sie sich auch auf der Website ein wenig erkunden und schauen, was wir gemacht haben. Sie können auch unseren Fortschritt von dem Projekt lesen, in dem Sie einfach unter dem grünen „Code“-Knopf auf „commits“ drücken. Entpacken Sie bitte dann den heruntergeladenen Zip-Ordner dahin, wo Sie es möchten.
3. Gehen Sie in den gerade entpackten Zip-Ordner (nicht der SFML-Ordner!) und gehen auf Projekt für VS und (wenn Sie Visual Studio schon installiert haben!) klicken Sie bitte doppelt auf die Solutions-Datei (Datei mit Namen „LasVegas.sln“!). Warten Sie bitte, bis das Projekt sich vollständig geöffnet hat.
4. Ruhe bewahren! Es wird höchstwahrscheinlich gleich auftauchen, dass Sie über 300 Fehler haben werden, aber dies passiert nur, weil Sie SFML noch nicht mit dem Projekt verlinkt haben.

ACHTUNG! Jeden Schritt sorgfältig durchlesen und nacheinander ausführen, da es sonst nicht funktionieren könnte.

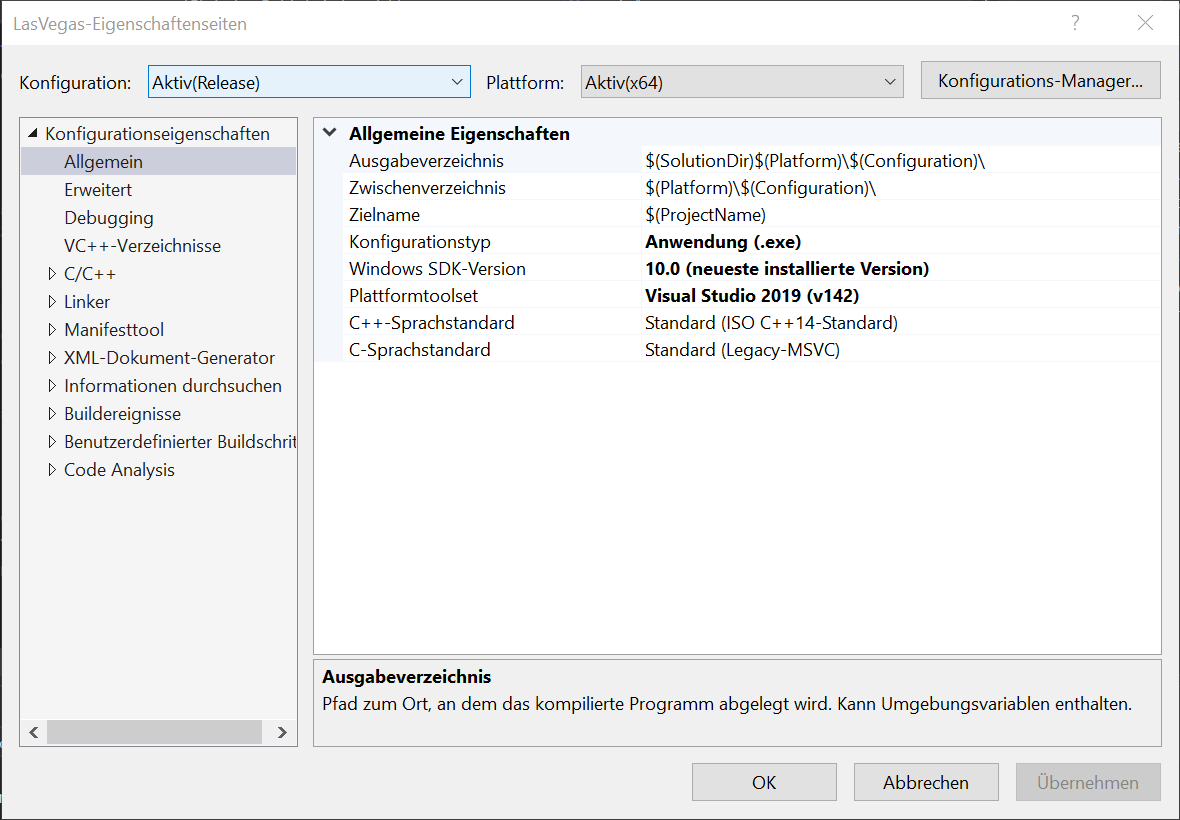
1. Gehen Sie oben in der Leiste auf Projekt/LasVegas-Eigenschaften. Dann müsste ein Fenster auftauchen, dass so aussieht. (Siehe Bild)

Abbildung Eigenschaftenseite

Achtung: Achten Sie darauf, dass ganz Oben als Konfiguration „Release“ und Plattform „x64“ steht! Ansonsten wird es nicht gehen! Ändern Sie es ggf. um!

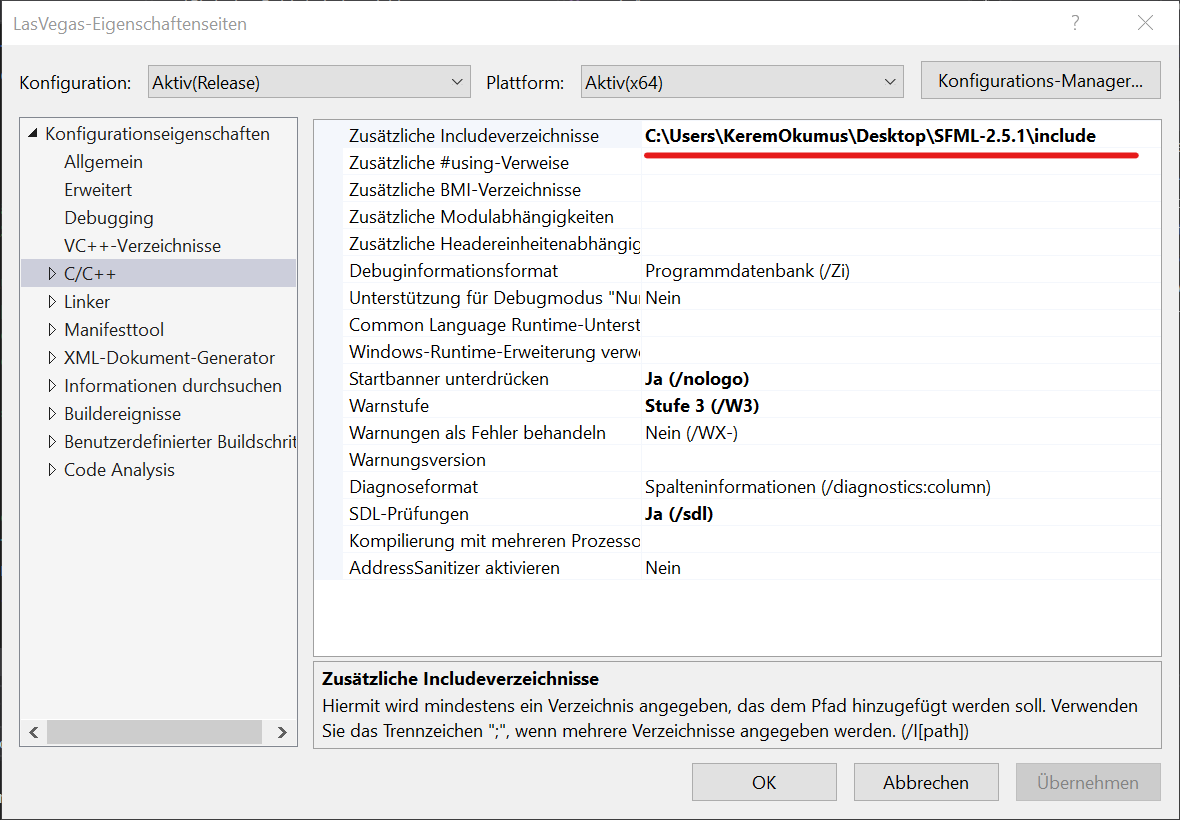
1. Gehen Sie dann auf C/C++ und ganz oben in der Liste rechts müsste jetzt „Zusätzliche Includeverzeichnisse“ stehen. (Siehe Bild 2)

Abbildung veränderter Includepfad

1. Suchen Sie jetzt den SFML-Ordner auf, gehen Sie in den include-Ordner.
2. Drücken Sie jetzt auf die Leiste oben. Kopieren Sie sich den Pfad, indem Sie auf STRG und C gleichzeitig drücken oder rechtsklick und dann auf Kopieren.
3. Gehen Sie anschließend wieder in Visual Studio rein, tippen Sie zweimal auf den vorhandenen Pfad und fügen Sie den neukopierten Pfad mit STRG und V oder rechtsklick und Einfügen ein. Das müsste dann jetzt so aussehen. (Siehe Bild 2)
4. Klicken Sie Links in der Liste auf Linker. Suchen Sie dann rechts in der Liste nach „Zusätzliche Bibliotheksverzeichnisse“.
5. Gehen Sie wieder zum SFML-Ordner und gehen Sie einen Ordner zurück, sodass oben in der Leiste nicht mehr „/include“ steht. Gehen Sie dann in den „lib“-Ordner und wiederholen Sie Schritt 8.
6. Gehen Sie wieder in Visual Studio und klicken Sie zweimal auf den vorhandenen Pfad (Siehe Schritt 10 für Hilfe) und fügen Sie den kopierten Link ein.
7. Stellen Sie sicher, dass oben als Konfiguration „Release“ und als Plattform „x64“ ausgewählt ist! Falls nicht, wählen Sie bitte diese aus und wiederholen Schritt 6. Bis 12.
8. Rechts unten auf Übernehmen drücken und dann auf OK.
9. Ändern Sie die Projektmappeneigenschaften auf „Release“ und „x64“ um. (Siehe Bild 3).
10. Klicken Sie oben auf Erstellen/Projektmappe erstellen.

Alle Fehler sollten jetzt langsam verschwinden, wenn nicht, warten Sie bitte ein paar Minuten. Wenn die Fehler nach 5 Minuten immer noch vorhanden sind, nochmal alles im Projekt-Eigenschaftenmenü überprüfen und falls nötig, alle Schritte wiederholen (sprich von 1 bis 16). Wenn’s immer noch nicht funktioniert, kontaktieren Sie bitte den Entwickler.

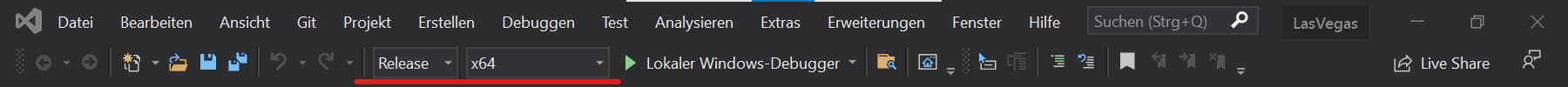


Abbildung Projektmappeneigenschaften

# Erklärung SFML-Programmierweise

SFML ist eine objektorientierte Bibliothek. Sprich man hat alles in einer Klasse geschrieben und man kann dann die Funktionen dieser Klasse mit einem Punkt aufrufen. Also, wenn ich jetzt von der Sounds-Klasse den Play-Funktion abrufen möchte, muss ich als Befehl Sounds.play(); schreiben.

Um zum Beispiel ein Bild anzeigen zu wollen, muss man das so machen:

sf::Texture golden\_nugget;

golden\_nugget.loadFromFile("res/Bilder/Assets/golden\_nugget.png");

golden\_nugget.setSmooth(true);

sf::Sprite golden\_nuggetsprite(golden\_nugget);

golden\_nuggetsprite.setPosition({ 20.0f, 350.0f }); spiel.draw(golden\_nuggetsprite);

Sieht für den Anfang viel aus, ist aber erstaunlich verständlich und einfach.

Fangen wir ganz von vorne an:

sf::Texture golden\_nugget;

Die erste Zeile erstellt den Objekt Texture mit dem Namen golden\_nugget;

golden\_nugget.loadFromFile("res/Bilder/Assets/golden\_nugget.png“);

In dieser Zeile geben wir dem Objekt golden\_nugget eine Eigenschaft. Und zwar, dass es ein Bild aus einer Datei laden soll und der Pfad zum Bild wird in der Klammer angegeben.

golden\_nugget.setSmooth(true);

In dieser Zeile stellen wir die Eigenschaft smooth auf true, um das Bild weicher zu zeichnen.

Jetzt wird es interessant. Wir werden jetzt einen Sprite erstellen, eine leere Fläche/Bilderrahmen, um die Textur darauf anzeigen zu lassen und werden diese Textur-Objekt in den Sprite setzen.

sf::Sprite golden\_nuggetsprite(golden\_nugget);

# Wichtige Dateien und deren Funktionen

Dieses Projekt umfasst eine Hauptdatei namens main.cpp und mehrere Headerdateien, die den Arbeitsfluss erleichtern. Außerdem, da dieses Projekt grafisch dargestellt wird, werden hier auch wichtige Bibliotheken wie openal32.dll, sfml-system.dll usw. verwendet, die die Nutzung der Bibliothek SFML (Simple and Fast Multimedia Library) ermöglichen.

## Main.cpp

Fangen wir mit dem Hauptprogramm an. Diese hat die meisten Zeilen Code und ist unser Hauptprogramm, mit den wichtigsten Funktionen. Bevor wir jedoch anfangen, werden wir als Erstes über das System des Spiels reden.

Da wir SFML als unsere primäre Bibliothek gewählt haben und wir dieses Spiel grafisch dargestellt haben, mussten Wir zur Vereinfachung dieses Spiel multithreaded erstellen.

Sprich, im Code gibt es Aufgaben und mehrere Threads(Arbeiter) arbeiten an ihren Aufgaben, anstatt dass ein Arbeiter alle Aufgaben gleichzeitig übernimmt, welches nicht möglich ist.

In diesem Spiel gibt es 3 Threads:

Wie Sie sehen können, ist das Projekt ziemlich komplexer als es scheinen mag, da auch vieles passieren muss. Zum Beispiel, wenn ein Spieler auf einen Würfel tippt, muss die Grafik aktualisiert werden, die dazu nötigen Variablen ergänzt oder verändert werden usw.

## main()

Wir befinden uns Jetzt ganz Unten im Code auf der Zeile 6705 in der Funktion main()

Die Main-Funktion ist das Erste, was in diesem Spiel gestartet wird. (Üblich wird das immer als Erstes im Programm gestartet)

Als erstes geben wir in der Konsole dem Benutzer bewusst, dass diese Konsole gar nicht üblich gebraucht wird und es nur als Fehlerausgabe fungieren wird.

Dann wird in derselben Funktion das Hauptmenü erstellt.

Gleichzeitig starten wir in der Funktion die Musik, die dann ewig läuft.

Der Nutzer kann nun entscheiden, was er machen möchte. Er kann ein neues Spiel starten, eine gespeicherte Sitzung laden, die Einstellungen aufrufen oder das Spiel einfach wieder beenden.

Gehen wir die möglichen Reaktionen und darauffolgende aufgerufene Funktionen durch:

Wenn der Spieler auf neues Spiel klickt, wird die Funktion neuesSpiel() aufgerufen.

Falls der Spieler auf Spiel laden klickt, ruft das Programm die Funktion dateienmanager() mit dem Wert false auf. (Wird in der Funktionenerklärung dafür erklärt, warum man false und true schreiben muss.)

Drückt der Spieler auf Einstellungen, startet das Programm die Einstellungen-Funktion auf.

Und falls der Spieler auf Beenden drückt, wird das Befehl exit() auf und beendet das Programm vollständig.

Schauen wir uns jedoch die anderen möglichen Funktionen an.

## neuesSpiel()

In dieser Funktion werden Variablen auf ein neues Spiel vorbereitet und zeigt dem Spieler das Spiel an.

Diese Funktion startet gleichzeitig am Anfang ein Thread namens zeichenthread und dieses startet die Funktion Spielzeichnung auf.

Das bedeutet, dass jetzt zwei Funktionen auf einmal laufen!

Bedingung:

Wenn davor kein Spiel geladen wurde, werden alle Scheine und Würfelwerte neu generiert.

Es werden auf Werte vom Spielzeichnung gewartet, und wenn es Signale bekommt, ändert diese Funktion Werte und Variablen und gibt dem Spielzeichnung Signale, dass es mit dem Spiel fortsetzen kann. Außerdem überprüft diese Funktion auch, ob ein Spieler gewonnen hat und gibt dann den Gewinner bekannt.

## Spielzeichnung()

In dieser Funktion wird das Hauptspiel gezeichnet. Anhand von Variablen, die andere Funktionen Werte zuweisen, zeichnet das Spiel die Informationen auf dem Bildschirm, sodass der Spieler sieht, was gerade welchen Wert hat. Dafür dass die Spielzeichnung in einem separaten Thread geschieht, ist es möglich, den Bildschirm ständig zu aktualisieren und man kann somit Verwirrungen und sonstige Wartezeiten verhindern. Außerdem ist das die größte Funktion im Programm, da es eine Größe von ungefähr 6000 Zeilen besitzt. Der Grund dafür ist, dass man für jeden Würfel, den man zeichnen muss, einen Platz reservieren muss und das geschieht mit ungefähr 9 Zeilen pro Würfel. Jedes Casino hat Platz für 30 Würfel und somit erreicht man eine große Zahl an Code.

## Gewinner(int spieler)

In dieser Funktion wird der Gewinner mit einem neuen Fenster bekannt gegeben und das aktuelle Spiel beendet. Man muss dabei einen Parameter angeben, und zwar der Wert Spieler. Dieser wird davor vom neuesSpiel()-Funktion übergeben. Dann hat man die Möglichkeit, ein neues Spiel zu starten, ein gespeichertes Spiel zu laden, ins Hauptmenü zu gelangen oder das Spiel zu beenden.

## para()

Diese Funktion ist dafür zuständig, dass nach jeder Runde jeder Spieler sein Betrag an Geld bekommt, den Würfeln zurückgesetzt werden, neue Geldscheine gelegt werden und man damit eine neue Runde starten kann.

## setzeSchrittZurück()

In dieser Funktion kann man einen Schritt, wenn man davor natürlich einen Würfel auf ein Feld gelegt hat, diesen Würfel wieder zurücknehmen.

Dies geschieht, indem man in der Funktion setzeWürfel() einen Würfel setzt und es dort dann die wichtigen Werte (Augenwert und Anzahl der Würfeln) in ein temporäres Array schreibt. Von hier aus werden dann den Augenwert mithilfe eines Switchs den Variablen zugeordnet und es werden Variablen wieder rückgängig gemacht.

## setzeWürfel(int spieler, int wert)

Mithilfe dieser Funktion werden den, von dem Spieler ausgewählten Würfeln, auf den jeweiligen Casino gesetzt. Dabei wird der ausgewählte Wert vom Spieler auf den Casinos(Casinovariablen) gesetzt. Am Ende werden die Anzahl und der Augenwert in eine temporäre Array gesetzt für das Zurücksetzen.

## Pausenmenu()

Diese Funktion dient dazu, dass man einfach das Spiel pausiert und ein Pausenmenü erscheint, damit der Spieler:

1. Den Spielstand speichern kann
2. Die Einstellungen aufrufen kann
3. Die Bedienungsanleitung aufrufen kann
4. Das Spiel beenden kann

## dateienmanager(bool speichern)

Mit dieser Funktion kann der Spieler seinen Spielstand speichern oder laden. Dies kann man entscheiden, indem man das bool speichern auf true setzt, damit man ein Spielstand speichern kann und auf false setzt, damit man ein Spielstand laden kann. Nebenbei kann man auch die Spielstände löschen, damit man diese ersetzen kann. Es wird einem Spieler nicht erlaubt, eine schon gespeicherte Datei zu überschreiben und es passiert dann gar nichts. Man muss sie also davor löschen.

## speicherdateilesen(int speichernummer)

Mithilfe dieser Funktion kann man eine Speicherdatei, die man davor als speichernummer angeben muss, ins Spiel laden. Diese Funktion setzt einfach nur die Werte der Datei in die Variablen und somit ist es sowie neuesSpiel, bloß dass die Variablen vorgegeben sind. Anschließend wird neuesSpiel gestartet aber davor wird noch ein bool namens spielgeladen auf true gesetzt, damit die Funktion neuesSpiel erkennt, dass er nicht neugenerieren soll.

## int gen()

Da wir eine Bibliothek verwenden, die nicht zur Standartbibliothek angehört, kann diese Bibliothek nicht mit rand() und rand2() klarkommen. Deswegen mussten wir einen neuen und auch standardisierten Zufallsgenerator schreiben. Diese wird von der C++-Version 11 unterstützt und ist legitim.

## Void Einstellungen()

In der Einstellungen funktion sind die Einstellungen für Schriftart und viele Variablen um die Oberfläche zu zeichnen enthalten.

## Header Button.h

In der Header Datei Button.h wird der Button durch Vorgabe der Schriftart, Geometrie, länge, breite und höhe gezeichnet. Außerdem wird geprüft wo sich der Mauscursor befindet.

## Header dateienmanager.h

In dieser Header-Datei werden allgemeine Arrays und Variablen wie z.b Casinogelder und Würfelanzahl deklariert . Außerdem liegt hier die Speicherfunktion die Variablen beschreibt oder löscht.

## Header Dice.h

Hier werden die Texturen der Würfel geladen und gezeichnet.

## Header infosystem.h

Ein Debugfenster läuft während des Spiels im Hintergrund, hier wird dieses gezeichnet und wichtige Werte während des Spielens ausgegeben. Wenn es zu Fehler kommen sollte, dann werden diese hier angezeigt.