Visual Studio 2019

Code faster. Work smarter. Create the future with the best-in-class IDE.

Download Visual Studio

- Community 2019
- Professional 2019
- Enterprise 2019

Visual Studio Installer

Before you get started, we need to set up a few things so that you can configure your installation.

To learn more about privacy, see the Microsoft Privacy Statement. By continuing, you agree to the Microsoft Software License Terms.

Continue

Installation details
- Visual Studio core editor
  - The Visual Studio core shell experience, including syntax-aware code editing, source code control and work item management.

Location
C:\Program Files (x86)\Microsoft Visual Studio 2019\Community

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the Readme.Md file in the accompanying license. By continuing, you also agree to those licenses.
On Windows, choosing 32 or 64-bit libraries should be based on which platform you want to compile for, not which OS you have. Indeed, you can perfectly compile and run a 32-bit program on a 64-bit Windows. So you'll most likely want to target 32-bit platforms, to have the largest possible audience. Choose 64-bit packages only if you have good reasons.

The compiler versions have to match 100%!
Here are links to the specific MinGW compiler versions used to build the provided packages:
TDM 5.1.0 (32-bit), MinGW Builds 7.3.0 (32-bit), MinGW Builds 7.3.0 (64-bit)

- Visual C++ 15 (2017) - 32-bit: [Download 16.3 MB]
- Visual C++ 14 (2015) - 32-bit: [Download 18.0 MB]
- Visual C++ 12 (2013) - 32-bit: [Download 18.3 MB]
- GCC 5.1.0 TDM (SJLJ) - Code::Blocks - 32-bit: [Download 14.1 MB]
- GCC 7.3.0 MinGW (GW2) - 32-bit: [Download 15.5 MB]
Configure your new project

Console App  

Project name

Timber

Location

D:\VS Projects\

Solution name

Timber

Place solution and project in the same directory

Create
openal32.dll
sfml-audio-2.dll
sfml-audio-d-2.dll
sfml-graphics-2.dll
sfml-graphics-d-2.dll
sfml-network-2.dll
sfml-network-d-2.dll
sfml-system-2.dll
sfml-system-d-2.dll
sfml-window-2.dll
sfml-window-d-2.dll

This is where we do our coding.
Internal Coordinates

(Origin = 0, 0)

\[ y = 0 \]

\[ x = 0 \]

increasing values of \( x \)

increasing values of \( y \)

Update all the game objects (move them, see if they collided, AI, etc.)

Draw all the game objects in their up-to-date positions

Respond to any screen touches from the user

Local Windows Debugger
Chapter 2: Variables, Operators, and Decisions – Animating Sprites
Chapter 3: C++ Strings and SFML Time – Player Input and HUD

SCORE = 0

PRESS ENTER TO START!

SCORE = 5206

SCORE = 0
Chapter 4: Loops, Arrays, Switches, Enumerations, and Functions – Implementing Game Mechanics
Chapter 5: Collisions, Sound, and End Conditions – Making the Game Playable
SCORE = 5

SQUISHED!!

SCORE = 10

FPS = 714.796
Chapter 6: Object-Oriented Programming – Starting the Pong Game

SCORE:0  LIVES:3

Chapter 7: Dynamic Collision Detection and Physics – Finishing the Pong Game

SCORE:0  LIVES:3
Chapter 8: SFML Views – Starting the Zombie Shooter Game

// The main game loop
while (window.isOpen()) {
}
```c
int main()
{
    // The main game loop
    while (window.isOpen())
    {
        // Game logic
    }
}
```
Chapter 9: C++ References, Sprite Sheets, and Vertex Arrays
Chapter 10: Pointers, the Standard Template Library, and Texture Management

Chapter 11: Collision Detection, Pickups, and Bullets
Chapter 12: Layering Views and Implementing the HUD
1. Increased rate of fire
2. Increased clip size (next reload)
3. Increased max health
4. Increased run speed
5. More and better health pickups
6. More and better ammo pickups
Chapter 13: Sound Effects, File I/O, and Finishing the Game
Chapter 14: Abstraction and Code Management – Making Better Use of OOP

HUD

Level:1

"Molten" effect background

Split screen coop

Level:1
Engine.h

private:
    input()
    update()
    draw()

public:
    run();

Engine.cpp

Engine::run()
{
    input();
    update();
    draw();
}

Input.cpp

Engine::input()
{
    // All the input code
}

Update.cpp

Engine::update()
{
    // All the updating code
}

Draw.cpp

Engine::draw()
{
    // All the drawing code
}
m_LeftView on top of m_BGLeftView

m_RightView on top of m_BGRightView

```cpp
#include "stdafx.h"
#include "Engine.h"

void Engine::draw() {
    // Code goes here
}
```
Chapter 15: Advanced OOP – Inheritance and Polymorphism
"Leap of faith" platforms

Goal
Chapter 18: Particle Systems and Shaders
**Screen**
- vector<UIPanel> m_Panels
- vector<InputHandler> m_Handlers
- View m_View

handleInput()
update()
draw()

**UIPanel1**
- vector<Buttons> m.Buttons
- View m_View

draw()

**InputHandler1**
- vector<Buttons> m.Buttons
- View m_View

handleInput()

**UIPanel2**
- vector<Buttons> m.Buttons
- View m_View

draw()

**InputHandler2**
- vector<Buttons> m.Buttons

handleInput()

**GameEngine**
- ScreenManager
  - m_ScreenManager

handleInput()
Update()
Draw()

**ScreenManager**
- String m_CurrentScreen = "Select"
- map<string, Screen> m_Screens

m_Screens[m_CurrentScreen]->handleInput()
m_Screens[m_CurrentScreen]->update()
m_Screens[m_CurrentScreen]->draw()

SwitchScreen(string nextScreen)
Screen

```cpp
update()
{
    // loop through all game objects
    ...
}

draw()
{
    // loop through all game objects
    ...
}
```

GameObject

```cpp
vector<Component>

- UpdateComponent
- GraphicsComponent
- ColliderComponent
- TransformComponent
```
SPACE INVADERS ++

Score: 0  Lives: 3  Wave: 7847

Score: 0  Lives: 3  Wave: 2

GAME OVER!
Chapter 21: File I/O and the Game Object Factory
Chapter 22: Game Objects and Building a Game

```cpp
#include "GameEngine.h"

int main()
{
    GameEngine m_GameEngine;
    m_GameEngine.run();
    return 0;
}
```