5½

A Case Study

CS/COE 0449
Introduction to
Systems Software

wilkie

Spring 2019/2020

History and Overview

I swear C is not really as hellish as this makes it seem.

Basic information

- First published as a mail-order shareware title in 1993 by Id Software; published by Apogee.
- A gritty-for-its-era early first-person action video game eventually defining the First-Person-Shooter genre (FPS)
- You play as a lone space marine in a survival-horror setting, albeit cartoony by today's standards, attempting to get to the exit of each level.

Yes, it was distributed on save icons

- DOOM originally came via mail-order purchase on four high-density floppy disks, a relatively large number at the time.
  - 1.44MB (or 1.41MiB) per disk is still quite the constraint.
  - Some modern websites download more data just to load their front page.
  - Yes. I have… every single version they ever released. On disks. Shut up.

An evergreen controversy

- **Although cartoony today, the game's violent aesthetic stood alone upon release causing a stir still reverberating today.**
  - The violence and satanic depictions throughout the game caused a stir among more conservative crowds.
  - The game was famously used by now-disgraced lawyer and activist Jack Thompson in his effort to highlight a possible (likely slightly true) correlation between games and real-world violent acts.
  - It was also infamously referenced by at least one of the perpetrators of the Columbine school shooting who was an avid player of the game.
    - It was one of the most popular games of its time… so that's not saying much.

Me, when I get a migraine

Indeed, we have many good modern ports!

I don't have a real good guess at how many people are going to be playing with this, but if significant projects are undertaken, it would be cool to see a level of community cooperation. I know that most early projects are going to be rough hacks done in isolation, but I would be very pleased to see a coordinated 'net release of an improved, backwards compatable* version of DOOM on multiple platforms next year.

Have fun.

John Carmack
12-23-97

* John's typo, not mine. Spell-checking cost extra back then.

Free and Open Source: “It Runs Doom”

- The DOOM source code has long been available to anyone since late 1997 and under an aggressively open license (GPL) since 1999.
  - Mirrored on GitHub: https://github.com/id-Software/DOOM/tree/master/linuxdoom-1.10
  - The modern port is GZDoom: https://zdoom.org/downloads

- The availability has allowed it to be studied by academics such as myself.
  - And it has been ported to ATMs and such because why not. “It Runs Doom”
  - It lives forever, essentially, as is the power of the open-source community.

Doom running on a (decommissioned) bank ATM

The Design

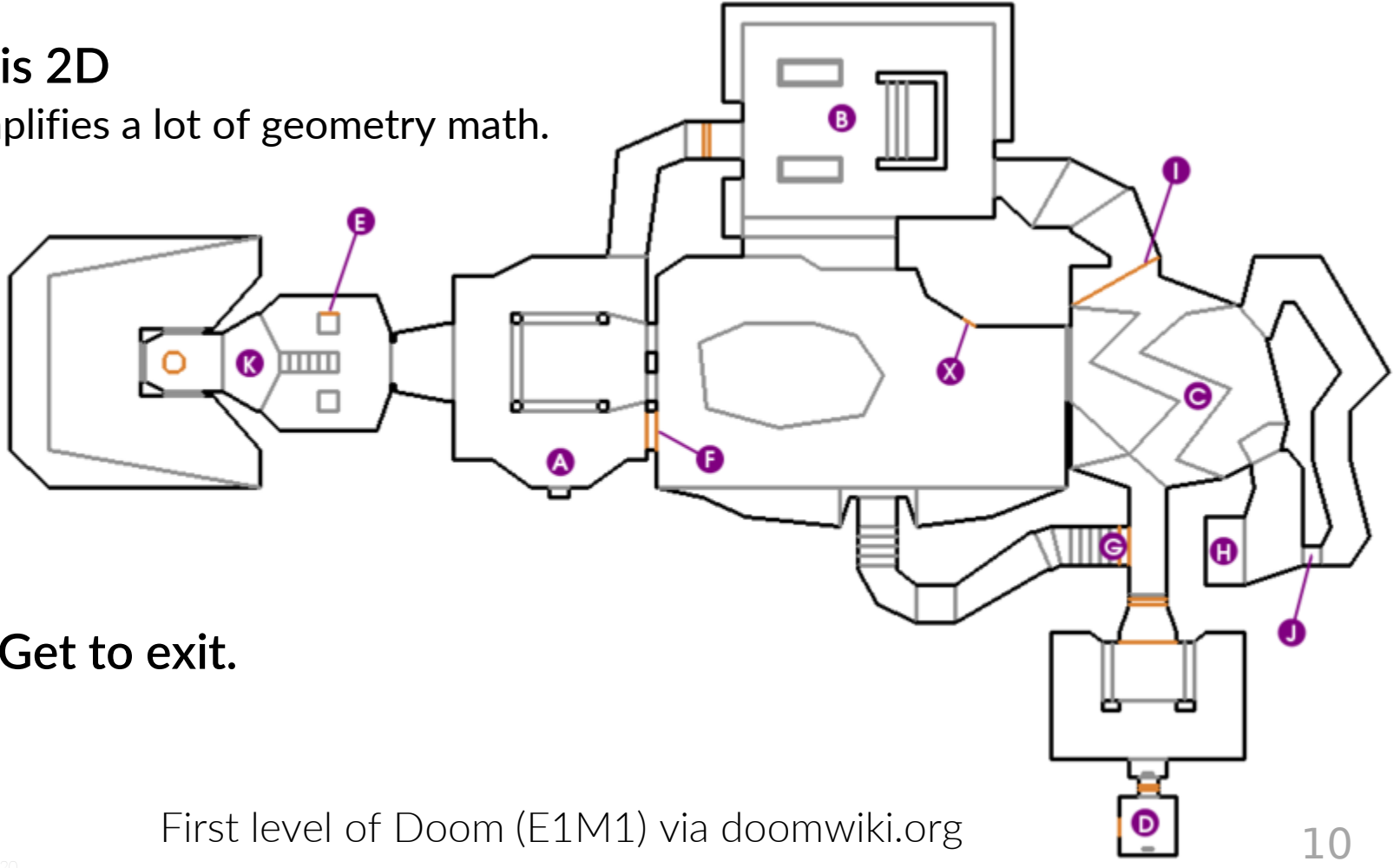
A multi-dimensional illusion in a game about teleporter mishaps. Fun!

The game, at a high-level

- A faux-3D textured environment.
  - Controlled movement only happens on X and Y axis (That is, no jumping.)
  - No slopes. All walls come to right angles with floors and ceilings.
- Fixed angle of view. Cannot look up or down or tilt the “camera.”
  - However, you can ascend vertically via stairs or simple elevators.
- Monsters populate each level and have minor intelligence to follow you by sight or sound and attack you or other monsters.
  - When monsters are struck by another’s projectile, they start attacking each other. Called “monster infighting”.
- Items populate the level and are picked up by colliding with them.
  - These include common genre tropes: ammo, health, armor.

Faux-3D Simplicity

- Level is 2D
  - Simplifies a lot of geometry math.



- Goal: Get to exit.

First level of Doom (E1M1) via doomwiki.org

Layout and Conventions

This showcases some weird C styling generally for greater speed/portability.

Coding Style

```
//
// A_TroopAttack
//
void A_TroopAttack (mobj_t* actor)
{
    int damage;

    if (!actor->target)
        return;

    A_FaceTarget (actor);
    if (P_CheckMeleeRange (actor))
    {
        S_StartSound (actor, sfx_claw);
        damage = (P_Random()%8+1)*3;
        P_DamageMobj (actor->target, actor, actor, damage);
        return;
    }

    // launch a missile
    P_SpawnMissile (actor, actor->target, MT_TROOPSHOT);
}
```

- **UpperCase function names.**
  - `A_` prefix denotes subsystem or category.
  - A good strategy to mitigate C's lack of classes or namespaces in large projects!
- **Allman/BSD style**
  - Braces on their own line.
- **ANSI C (C89) dialect.**
  - Variables declared at top.
- **Passes arguments by-ref**
  - Avoids return values often.
  - Prefers pointers to structs.

Header files and global variables…

doomdef.h

```
// Game mode handling - identify IWAD version
//  to handle IWAD dependend animations etc.
typedef enum
{
  shareware,    // DOOM 1 shareware, E1, M9
  registered,   // DOOM 1 registered, E3, M27
  commercial,   // DOOM 2 retail, E1 M34 retail,
                // DOOM 1 retail, E4, M36
  indetermined  // Well, no IWAD found.
} GameMode_t;
```

doomstat.h

```
// ---------------------------------------------
// Game Mode - identify IWAD as shareware,
// retail etc.
extern GameMode_t  gamemode; // Says there is a
                             // gamemode variable
                             // somewhere...
```

doomstat.c

```
GameMode_t  gamemode;  // The ACTUAL gamemode variable
```

- **Separating C code into multiple files:**
  - `*.h` files contain types/enums and function declarations.
  - `*.c` files will contain function implementations.

- **The `extern` says "Don't create this. It exists somewhere else."**
  - It makes a global available to other C files. They just `#include` the header.
  - You declare it once (without `extern`) in the corresponding C file.
  - Therefore, each C file can share data.

Header files and global variables…

g_game.h

```
//
// GAME
//
void G_DeathMatchSpawnPlayer (int playernum);

void G_InitNew (skill_t skill, int episode, int map);

// Can be called by the startup code or M_Responder,
// calls P_SetupLevel or W_EnterWorld.
void G_LoadGame (char* name);

void G_DoLoadGame (void);

// Called by M_Responder.
void G_SaveGame (int slot, char* description);

void G_ExitLevel (void);
void G_SecretExitLevel (void);

void G_WorldDone (void);
```

- **Separating C code into multiple files:**
  - `*.h` files contain types/enums and function declarations.
  - `*.c` files will contain function implementations.
- **You can declare functions without implementations.**
  - Typically for header files.
  - Implement them in their `*.c` file.
    - `g_game.c` in this case
  - Including this header file will allow use of this function.

Header files and global variables…

g_game.c

```
// wilkie's note: remember boolean isn't defined by C!
boolean       secretexit;
// wilkie's note: gameaction_t defined elsewhere
gameaction_t  gameaction;

void G_ExitLevel (void)
{
    secretexit = false;
    gameaction = ga_completed;
}

// Here's for the german edition.
void G_SecretExitLevel (void)
{
    // IF NO WOLF3D LEVELS, NO SECRET EXIT!
    if ( (gamemode == commercial)
      && (W_CheckNumForName("map31")<0))
        secretexit = false;
    else
        secretexit = true;
    gameaction = ga_completed;
}
```

- Here we see the implementations of some of the functions.
  - Doom likes its private global data!
  - No file in the Doom source code defines `secretexit` as `extern` so only this file can use it!

- Cultural artifacts become code.
  - Code reflects culture.
  - The secret levels in Doom are in the style of their earlier WWII-era game, Wolfenstein 3D.
  - They need to special case some code because German law does not allow Nazi imagery.

Bugs happen to all of us…

d_main.c

```
#include "doomdef.h"
#include "doomstat.h“
#include <unistd.h> // for 'access'
#include <stdlib.h> // for 'getenv'
#include <stdio.h>  // for 'sprintf'

// Checks availability of IWAD files by name,
// to determine whether registered/commercial features
// should be executed (notably loading PWAD's).
void IdentifyVersion (void) {
    char* doom1wad;
    char* plutoniawad;
    char* doomwaddir = getenv("DOOMWADDIR");

    if (!doomwaddir)
        doomwaddir = ".";

    // Shareware.
    doom1wad = malloc(strlen(doomwaddir)+1+9+1);
    sprintf(doom1wad, "%s/doom1.wad", doomwaddir);
    // Bug, dear Shawn.
    // Insufficient malloc, caused spurious realloc errors.
    plutoniawad = malloc(strlen(doomwaddir)+1+/*9*/12+1);
    sprintf(plutoniawad, "%s/plutonia.wad", doomwaddir);

    if ( !access (plutoniawad, R_OK ) )
    {
      gamemode = commercial;
      D_AddFile (plutoniawad);
      return;
    }

    if ( !access (doom1wad,R_OK) )
    {
      gamemode = shareware;
      D_AddFile (doom1wad);
      return;
    }

}
```

sprintf **is like** printf, **but prints to a buffer.**

checks if file exists (0 on success)

sets the gamemode **global**

- Apparently, somebody corrected Shawn Green's copy-and-paste of malloc.
  - It was not allocating enough space for the string copy!
    - Try not to copy and paste!!

The culture of bug hunting

- A person that studies insects is called an entomologist.
  - However, the first documented case of a real-life insect causing an error in a computer was a moth. A story chronicled by Grace Hopper.
  - Somebody who studies moths/butterflies is called a lepidopterist.
  - (This will not be on the exam)

- Ok… ok… the word bug in engineering is hundreds of years old.
  - We're not sure where it actually comes from.
  - However, today, we call the modern bug hunter a "speedrunner."

- It's a bit of an artform of its own these days.

The fear of moths is called Mottephobia.

Looking at things at another angle…

4shockblast, E1M1 UV/Pacifist, 0:08s - Record set in 2019 after 22 years unbroken
https://www.youtube.com/watch?v=1UkeFwJ-yHI

- **Consider:**
  - In what weird way is the player moving?
  - Do you believe this to be intentional? Why might that be faster?

Speedrunning

Gotta go fast. Speedrunning is just a mistake made beautiful.

Going fast is not always “straight-forward”

- If you notice from the video, the speedrunner is running at an angle.
  - This allows them to move at a higher speed.
- To figure out why this works, we can consult the source code.
- First, some information:
  - Doom lets you move forward and backward and turn left and right.
  - There is also the ability to use keys to strafe left and right instead.
  - Strafing moves you side by side (like a crabwalk)
- Let’s see how Doom implements movement!

We're drifting in two directions

p_user.c

```
boolean onground;

//
// P_MovePlayer
//
void P_MovePlayer (player_t* player)
{
    ticcmd_t*   cmd;

    cmd = &player->cmd;

    player->mo->angle += (cmd->angleturn<<16);

    // Do not let the player control movement
    //  if not onground.
    onground = (player->mo->z <= player->mo->floorz);

    if (cmd->forwardmove && onground)
        P_Thrust (player, player->mo->angle, cmd->forwardmove*2048);

    if (cmd->sidemove && onground)
        P_Thrust (player, player->mo->angle-ANG90, cmd->sidemove*2048);
}
```

yes, double dereferences are a thing!

The 2048 is a scalar; to avoid floating point!

- First, it makes sure the player is on the ground.
- If so, and you are pressing "forward" move the player forward!
- If so, and you are pressing "strafe" (sidemove), move the player sideways!
- Wait! What if both happen?

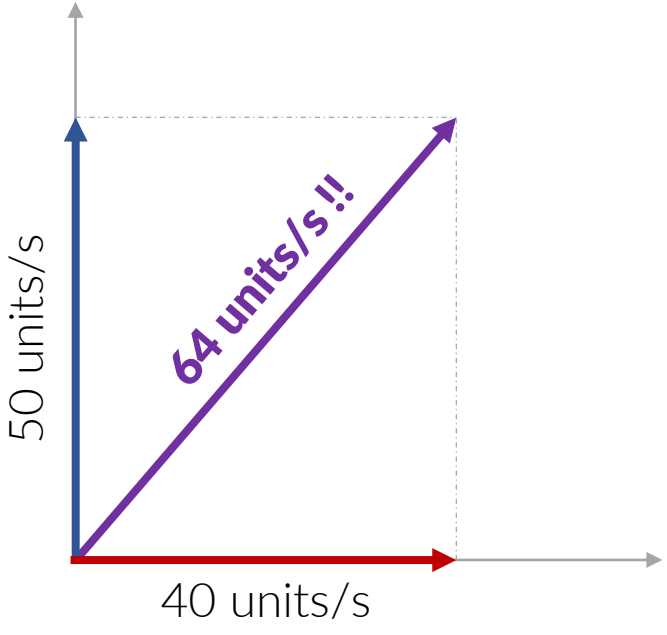
What's the Hypotenuse

- Holding both forward and strafe keys down moves the player.

- These are both independent movement.
  - The `cmd->forwardmove` is, at most, 50
  - The `cmd->sidemove` is, at most, 40
  - (These are set in `g_game.c`)

- Therefore, you are moving:
  - 50 units forward
  - 40 units sideways
    - Hence, we call this trick "strafe 40" or SR-40

- For a total speed of:
  - 64 units/second



Can we do better?

- Doom lets you use a single key to strafe left and right.
- It also lets you press a toggle to repurpose the left and right keys.
  - So, if you press this “strafe toggle” key, left and right no longer turn you.
  - They strafe you.

- Let’s look at the Doom code for how this is implemented.

Let's go even faster!

g_game.c

```
#define MAXPLMOVE 50

void G_BuildTiccmd (ticcmd_t* cmd) {
    boolean strafe;
    int     speed;
    int     forward;
    int     side;

    strafe = gamekeydown[key_strafe];
    speed = gamekeydown[key_speed];   // The "run" button

    forward = side = 0;

    // let movement keys cancel each other out
    if (strafe) {
        if (gamekeydown[key_right]) {
            side += sidemove[speed];  // This is 40
        }
        if (gamekeydown[key_left]) {
            side -= sidemove[speed];
        }
    }

    if (gamekeydown[key_up])
        forward += forwardmove[speed];
    if (gamekeydown[key_down])
        forward -= forwardmove[speed];

    if (gamekeydown[key_straferight])
        side += sidemove[speed];  // This is 40
    if (gamekeydown[key_strafeleft])
        side -= sidemove[speed];

    if (forward > MAXPLMOVE)
        forward = MAXPLMOVE;
    else if (forward < -MAXPLMOVE)
        forward = -MAXPLMOVE;

    if (side > MAXPLMOVE)
        side = MAXPLMOVE; // What is this, at most?
    else if (side < -MAXPLMOVE)
        side = -MAXPLMOVE;

    cmd->forwardmove += forward;
    cmd->sidemove += side;
}
```

Whether or not the "strafe toggle" is on

If so, "right" strafes right.

I removed the else which turned the player

The normal strafe key.

OOPS

OOPS

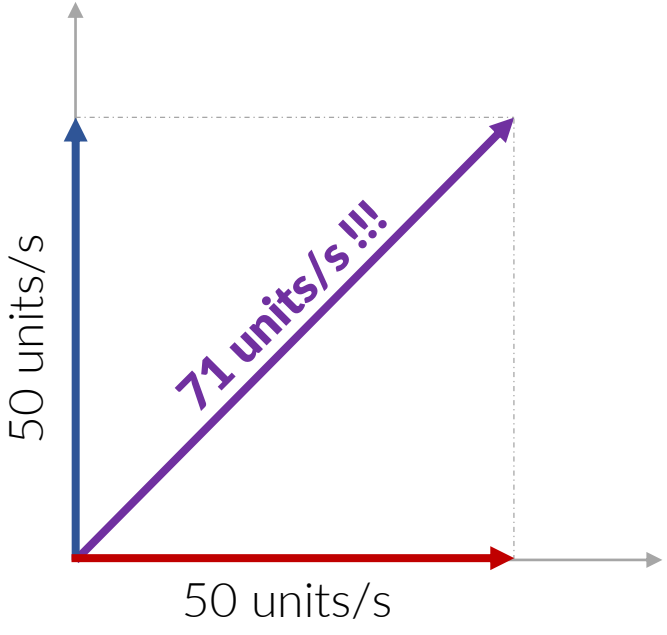
What's the Hypotenuse (revisited)

- Forward and both types of strafe moves the player very quickly.

- These are both independent movement.
  - The `cmd->forwardmove` is, at most, 50
  - The `cmd->sidemove` is now, mistakenly, 50
  - (These are set in `g_game.c`)

- Therefore, you are moving:
  - 50 units forward
  - 50 units sideways
    - Hence, we call this trick "strafe 50" or SR-50

- For a total speed of:
  - 71 units/second



Out-running Rockets: GOING FASTER STILL

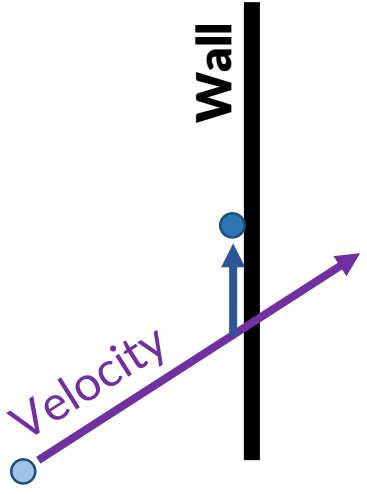
The WALLRUN.LMP demo file from E2M8.
https://www.youtube.com/watch?v=xJ7PgOJKbEk

The "Wall Run"

- The reason behind this behavior was unknown for quite some time.
- However, the wall-run has to do with improper collision handling.

- When you hit a wall, you slide along it.
  - There is some friction (you should SLOW down)
  - And some code to carry your momentum along the angle of the wall. (a vector projection)

- However, for certain walls it speeds up.
  - Going exactly north along a wall.
  - Going exactly east along a wall.



Intended Behavior

Let's go even faster!

- This function moves objects given their momentum.

p_mobj.c

```
#define MAXMOVE 30
void P_XYMovement (mobj_t* mo) {
    fixed_t   ptryx; // Position-try-x,
    fixed_t   ptryy; //  "-try-y: The possible new position
    fixed_t   xmove; // Amount to move along x
    fixed_t   ymove; // Amount to move along y

    if (mo->momx > MAXMOVE)
        mo->momx = MAXMOVE;
    else if (mo->momx < -MAXMOVE)
        mo->momx = -MAXMOVE;

    if (mo->momy > MAXMOVE)
        mo->momy = MAXMOVE;
    else if (mo->momy < -MAXMOVE)
        mo->momy = -MAXMOVE;

    xmove = mo->momx;
    ymove = mo->momy;
```

When moving quickly, it moves half-way twice.

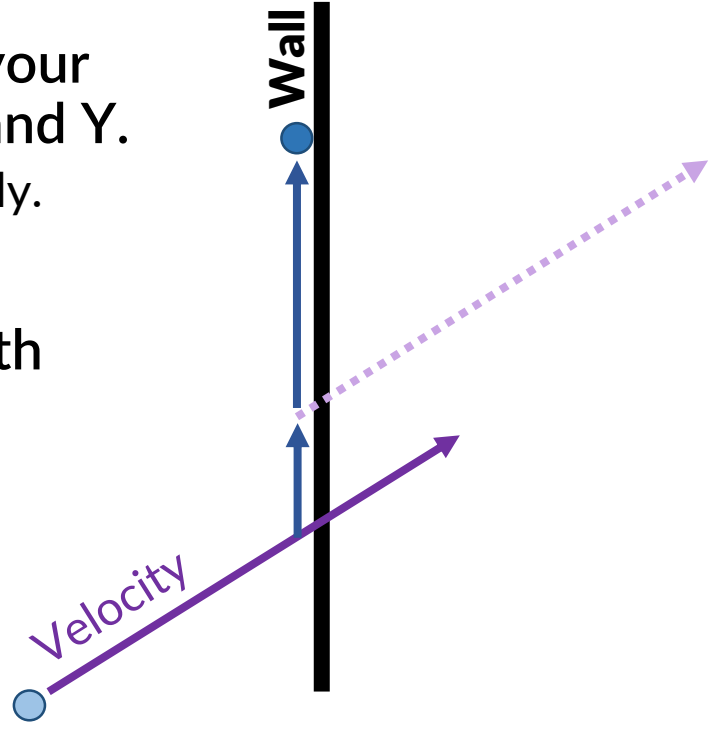
However, the slide-along-walls function did not get the memo!

```
    do {
        if (xmove > MAXMOVE/2 || ymove > MAXMOVE/2) {
            ptryx = mo->x + xmove/2; // Move half-way
            ptryy = mo->y + ymove/2;
            xmove >>= 1; // Divides by two,
            ymove >>= 1; // we move this much next time
        }
        else {
            ptryx = mo->x + xmove; // Move the
            ptryy = mo->y + ymove; //   whole way.
            xmove = ymove = 0; // We are done moving
        }

        if (!P_TryMove (mo, ptryx, ptryy)) {
            // blocked move
            if (mo->player) {
                // try to slide along it
                P_SlideMove (mo);
            }
            else
                mo->momx = mo->momy = 0;
        }
    } while (xmove || ymove);
}
```

The “Wall Run” visualized

- When going North or East, you’ve maximized your X or Y momentum respectively.
- Going slightly north-east also works, but your momentum is, then, slightly split along X and Y.
  - This code looks at each momentum individually.
- When the slide function is called twice with your original momentum… it causes…
  … well … dramatic results.
- We call this an “Oops” in the trade.



Unintended Behavior

Hmm, only North? Only East? Oh… Oops!

- This function moves objects given their momentum.

p_mobj.c

```
#define MAXMOVE 30
void P_XYMovement (mobj_t* mo) {
    fixed_t   ptryx; // Position-try-x,
    fixed_t   ptryy; //  "-try-y: The possible new position
    fixed_t   xmove; // Amount to move along x
    fixed_t   ymove; // Amount to move along y

    if (mo->momx > MAXMOVE)
        mo->momx = MAXMOVE;
    else if (mo->momx < -MAXMOVE)
        mo->momx = -MAXMOVE;

    if (mo->momy > MAXMOVE)
        mo->momy = MAXMOVE;
    else if (mo->momy < -MAXMOVE)
        mo->momy = -MAXMOVE;

    xmove = mo->momx;
    ymove = mo->momy;
```

Momentum can be negative, yet…

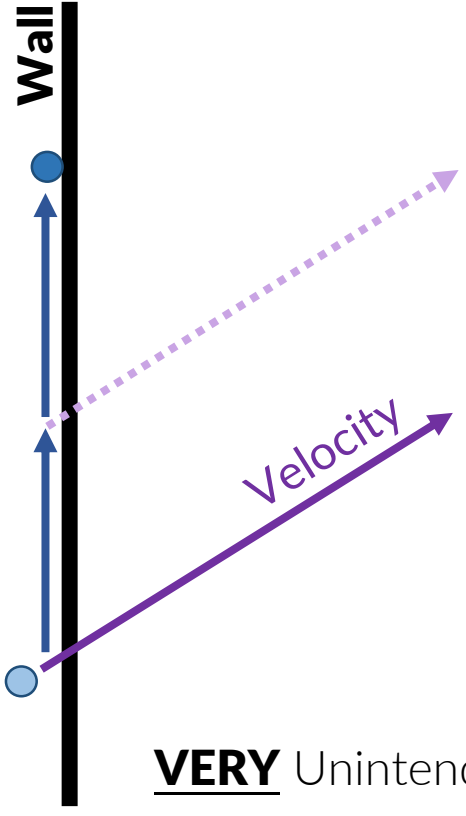
How many times does this get called when momentum is VERY positive? And, VERY negative?

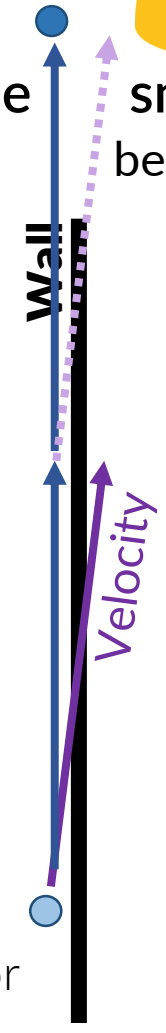
```
    do {
        if (xmove > MAXMOVE/2 || ymove > MAXMOVE/2) {
            ptryx = mo->x + xmove/2; // Move half-way
            ptryy = mo->y + ymove/2;
            xmove >>= 1; // Divides by two,
            ymove >>= 1; // we move this much next time
        }
        else {
            ptryx = mo->x + xmove; // Move the
            ptryy = mo->y + ymove; //   whole way.
            xmove = ymove = 0; // We are done moving
        }

        if (!P_TryMove (mo, ptryx, ptryy)) {
            // blocked move
            if (mo->player) {
                // try to slide along it
                P_SlideMove (mo);
            }
            else
                mo->momx = mo->momy = 0;
        }
    } while (xmove || ymove);
}
```

The “Wall Run” visualized

- The closer you are to the wall and the smaller your angle…
  - The more like Sonic the Hedgehog you become.





VERY Unintended Behavior

Here I am at the end…
Ruining my slides…

I can’t believe this is 10 years old.

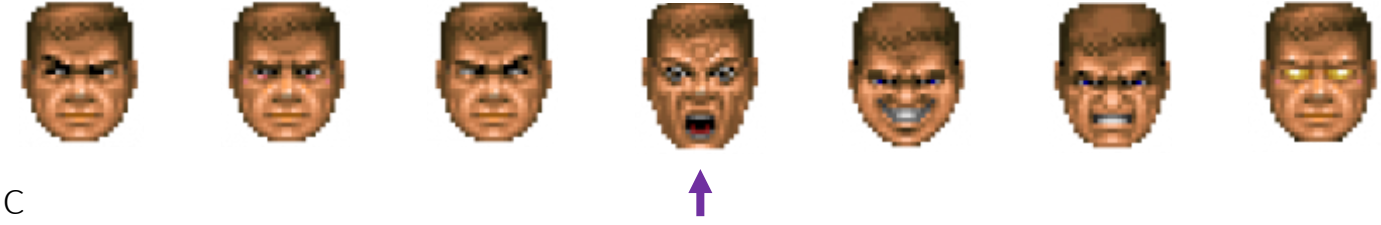
Closing Remarks

“I have not failed. I've just found 10,000 ways that won't work.”

— An elephant murderer talking about other people’s work

Your turn!

- **With your knowledge of C, you unlock a lot of potential.**
  - You can figure out quite a few interesting things.
- **The "face" depicting the character will change based on the action.**
  - However, this face only appeared in very odd circumstances.
  - Can you spot the bug? (Hint: you can both gain and lose health simultaneously.)



st_stuff.c

```
if (plyr->damagecount) {          // ST_MUCHPAIN is 20
  if (plyr->health - st_oldhealth > ST_MUCHPAIN) {
    // Selects the VERY HURT character portrait:
    st_faceindex = 3;
  }
}
```

That face is kinda scary, so it's probably for the best.

Are we DOOMed to fail?

- **In the end, I want to pass off some key knowledge nuggets:**
  - Perfect code is an illusion used to sell textbooks.
- **We have gained more from these mistakes than if they weren't there: art, culture, and enjoyment.**
  - The speedrunning community finds art in our mistakes.
  - Fixing them does not improve the art, just hints at our vanity.
  - The art culture around failure is not about judgement, but exploration.
- **Do not come out of this course thinking perfection is the goal.**
  - Mistakes in C can be costly for kernels, etc… sure.
  - However, in your day-to-day life, know that often these small mistakes are certainly common at all levels of skill. They can be fixed. They can be fun.
- **So, have fun writing and reading code!**