MESSAGING IN C

FOR GAM 150 CLUB

Randy Gaul

Who am I?

- RTIS Sophomore – Randy Gaul
- C game as Freshman
- Tech director for Ancient Forest and Grumpy Monsters
- Made engine in C during summer before Sophomore year
  - AsciiEngine
- Love architecture with clean and powerful APIs

Lecture overview

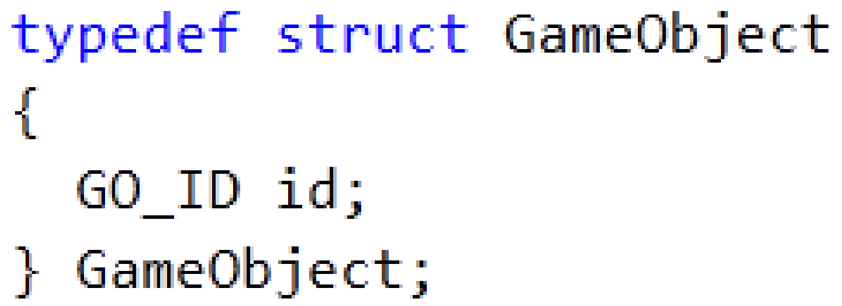
- Review Game Object Design
- The problem
- The solution: Messaging
- Messaging Implementation in C
- Questions?

Read the Game Object Design ppt

- It should be up on Moodle in the GAM 150 Club
- If not just email me, I’ll send it to you
  - r.gaul@digipen.edu
- Next few slides are refresher

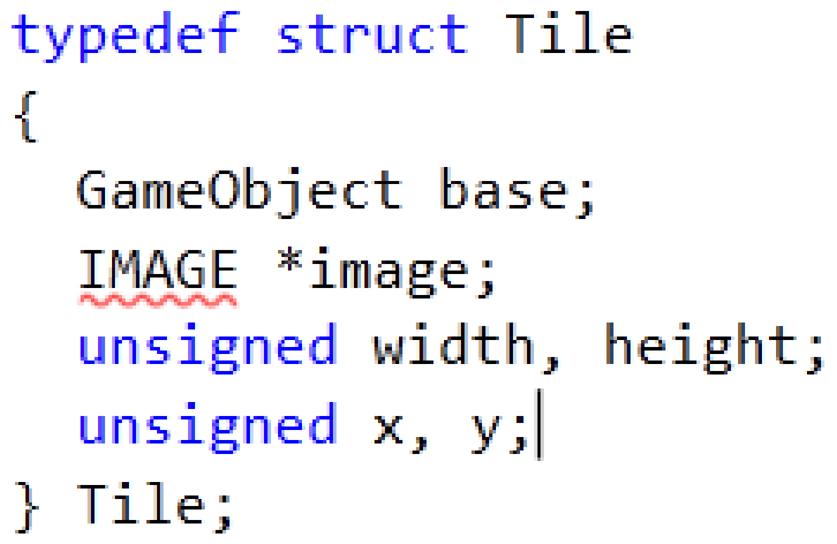
Inheritance in C - Refresher

- Here’s our game object structure



```c
typedef struct GameObject
{
  GO_ID id;
} GameObject;
```

- Here’s our Tile structure
  - Tile is a type of game object
  - Can have any number of types

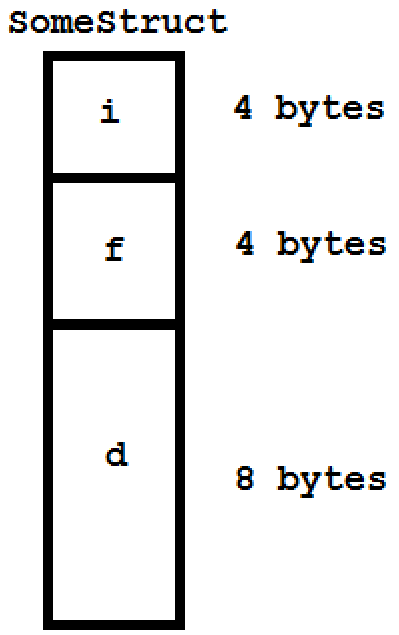


```c
typedef struct Tile
{
  GameObject base;
  IMAGE *image;
  unsigned width, height;
  unsigned x, y;
} Tile;
```

Inheritance in C - Refresher

- Why would you place a struct inside a struct?
- Here's a structure and memory diagram:

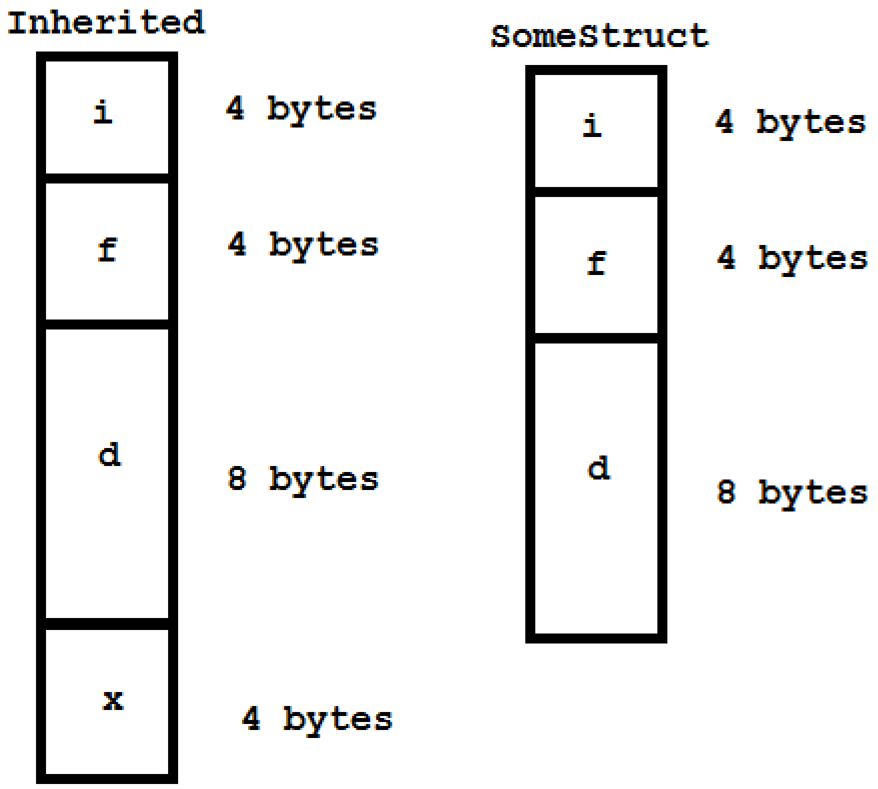
```c
typedef struct SomeStruct
{
  int i;
  float f;
  double d;
} SomeStruct;
```



Game Objects in C - Inheritance

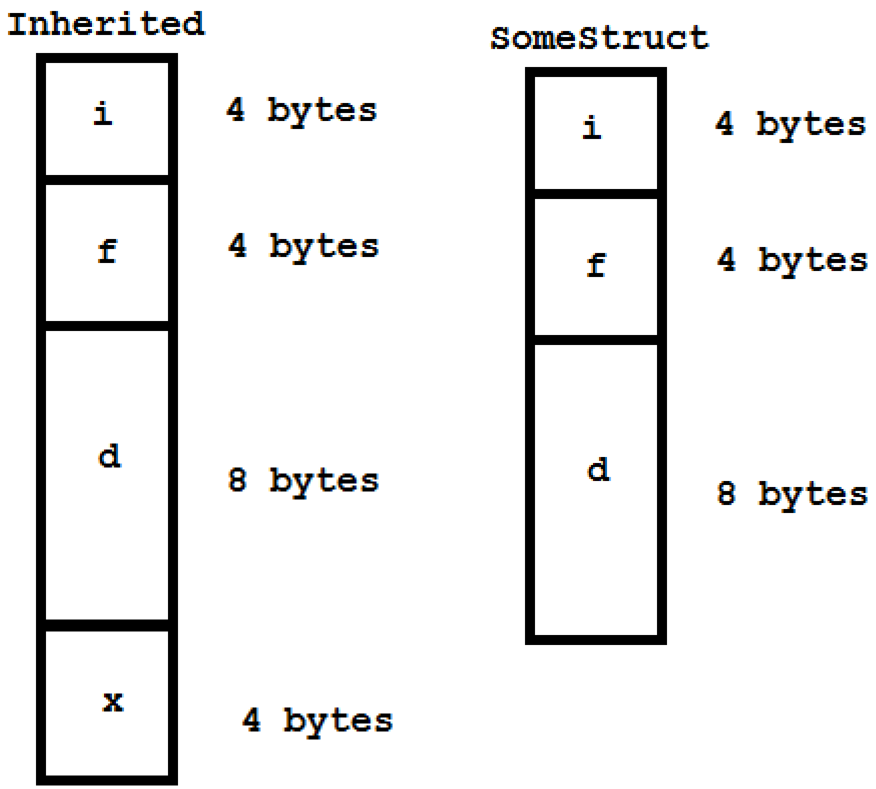
- Now take a look at a struct with inheritance

```
typedef struct inherited
{
   SomeStruct base;
   int x;
} inherited;
```



Inheritance in C - Refresher

- The top portion of inherited is the base
  - Can treat inherited memory as base
  - Can typecast inherited pointer to base pointer
- Generalized code
  - Function takes a base pointer, can pass an inherited pointer casted to base type



The Problem

- Imagine you’ve been using inheritance…
  - Made different kinds of objects
  - Each object type has a header
    - Player.h
    - Enemy1.h
    - Tile.h
- Realize it’s really annoying to…
  - Typecast base to inherited
  - Include file to call function
- Why is this annoying? Good question…

The Problem – Why it's annoying

- Lets assume we make a new object
  - Here's the header:

```
void RedEnemyInit   ( GameObject *self );
void RedEnemyDraw   ( GameObject *self );
void RedEnemyUpdate ( GameObject *self, float *dt );
void RedEnemyDestroy( GameObject *self );
```

- Now lets call a RedEnemy func from base:

```
GameObject *enemy;

((RedEnemy *)enemy)->FindNearestTarget( );
// Or the fancy way
CAST( enemy, RedEnemy )->FindNearestTarget( );
```

  - Cast macro:

```
#define CAST( PTR, TYPE ) \
  ((TYPE *)PTR)
```

The Problem – Why it's annoying

- In order to call function:

```
GameObject *enemy;

((RedEnemy *)enemy)->FindNearestTarget( );
// Or the fancy way
CAST( enemy, RedEnemy )->FindNearestTarget( );
```

- You must include file
  - RedEnemy.h

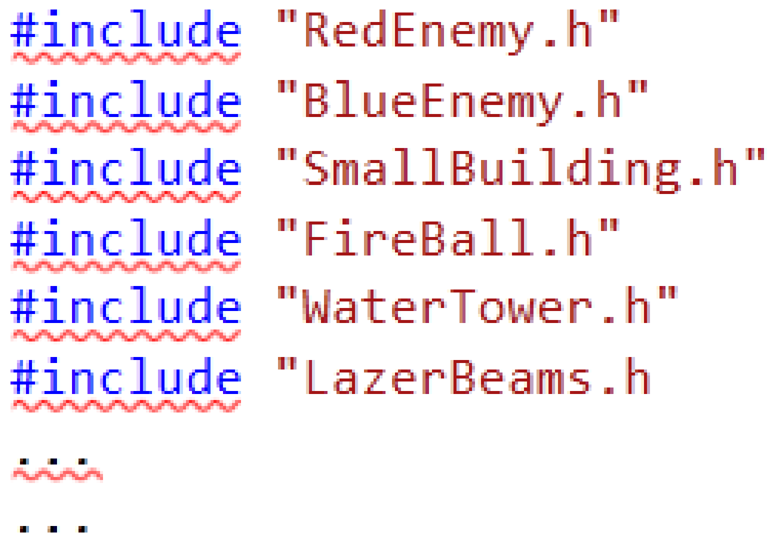
```
void RedEnemyInit   ( GameObject *self );
void RedEnemyDraw   ( GameObject *self );
void RedEnemyUpdate ( GameObject *self, float *dt );
void RedEnemyDestroy( GameObject *self );
```

The Problem – Why it’s annoying

- Imagine many objects interacting
  - Deal damage to one another
  - Run into each other
  - Follow patterns
  - Pathfinding

The Problem – Why it’s annoying

- Imagine many objects interacting
  - Deal damage to one another
  - Run into each other
  - Follow patterns
  - Pathfinding



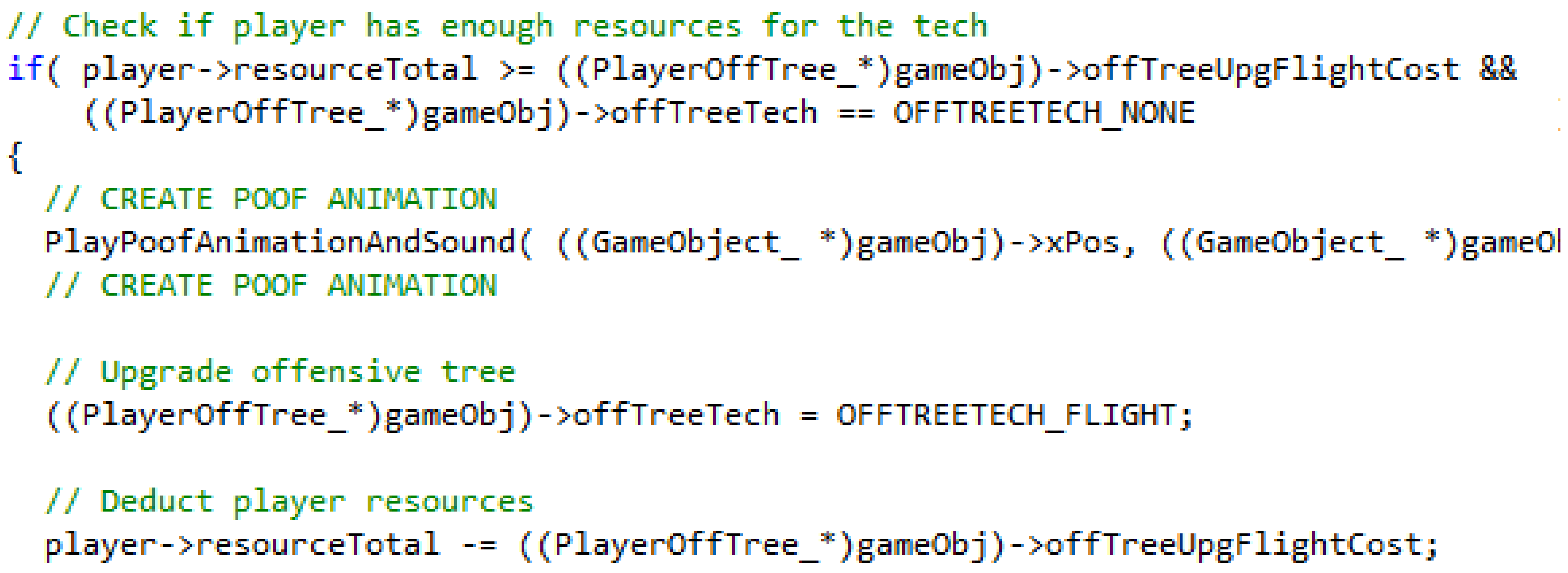
```
#include "RedEnemy.h"
#include "BlueEnemy.h"
#include "SmallBuilding.h"
#include "FireBall.h"
#include "WaterTower.h"
#include "LazerBeams.h
...
...
```

The Problem – Why it's annoying

- Things will get hectic
  - What file do I need for this?
    - 3 minutes searching
  - What do I typecast to again?
    - Another 2 minutes
- Minutes add up!
  - Don't waste your minutes
- You want to spend time making things
  - Searching is distracting
    - Productivity down the drain

The Problem – Why it’s annoying

- Horrible ugly typecasting code
  - Excerpt from my GAM 150 game:



```
// Check if player has enough resources for the tech
if( player->resourceTotal >= ((PlayerOffTree_*)gameObj)->offTreeUpgFlightCost &&
    ((PlayerOffTree_*)gameObj)->offTreeTech == OFFTREETECH_NONE
{
  // CREATE POOF ANIMATION
  PlayPoofAnimationAndSound( ((GameObject_ *)gameObj)->xPos, ((GameObject_ *)gameOb
  // CREATE POOF ANIMATION

  // Upgrade offensive tree
  ((PlayerOffTree_*)gameObj)->offTreeTech = OFFTREETECH_FLIGHT;

  // Deduct player resources
  player->resourceTotal -= ((PlayerOffTree_*)gameObj)->offTreeUpgFlightCost;
```

The Solution

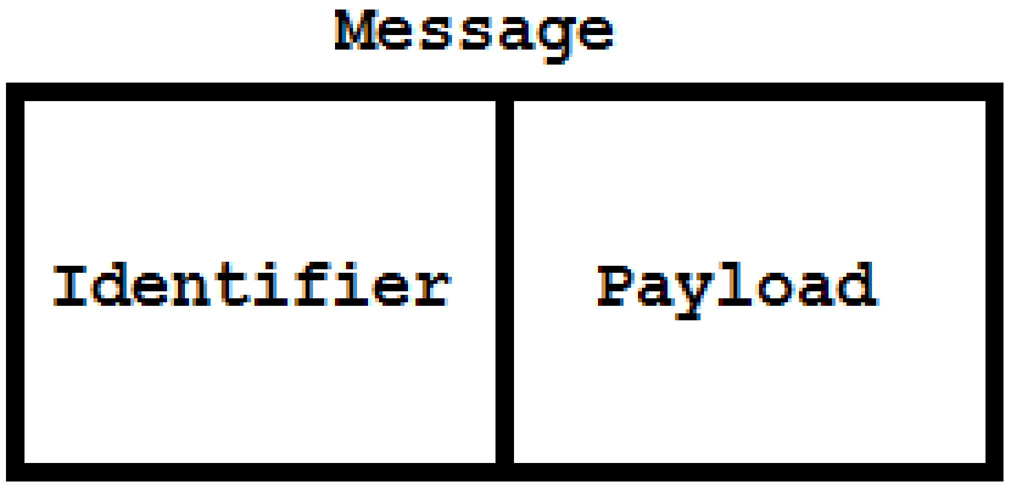
- Needs:
  - Call function on inherited
  - No hectic searching
  - Minimal file inclusion
  - Avoid typecasting GameObject *

The Solution - Messaging

- Send information from one place to another
- Done with a special function call

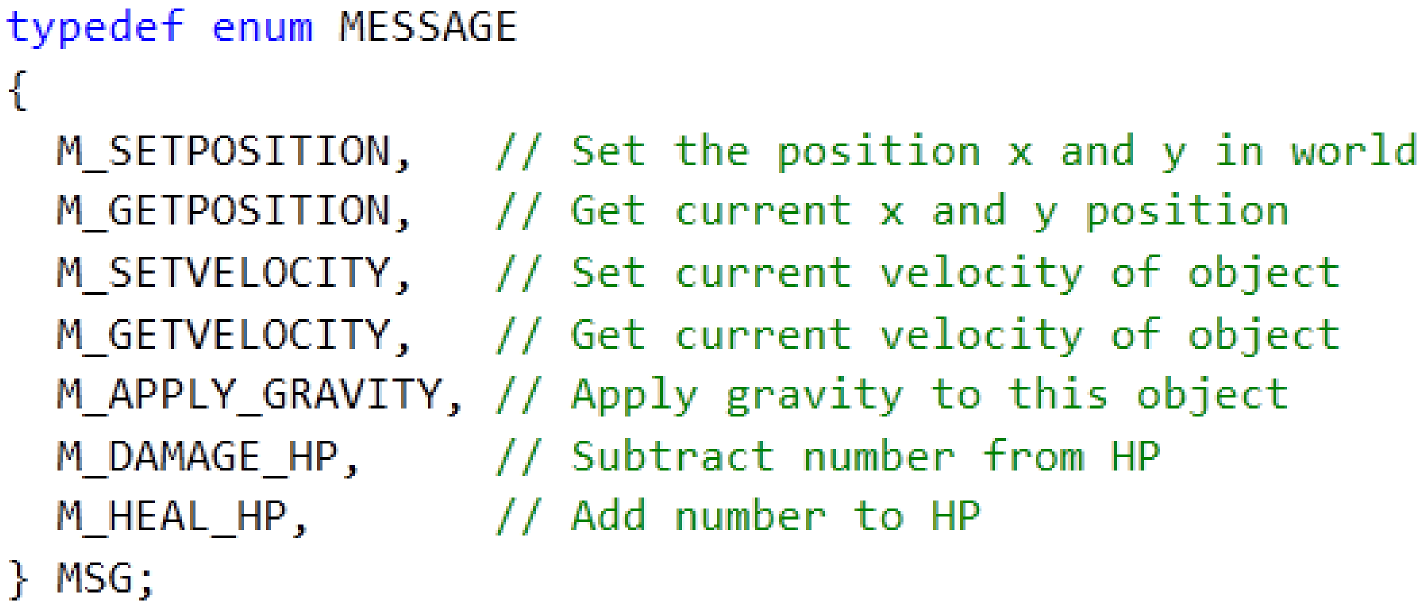
The Solution - Messaging

- Diagram of a message



Messaging

- Message identification
  - Enumeration – in a header file!

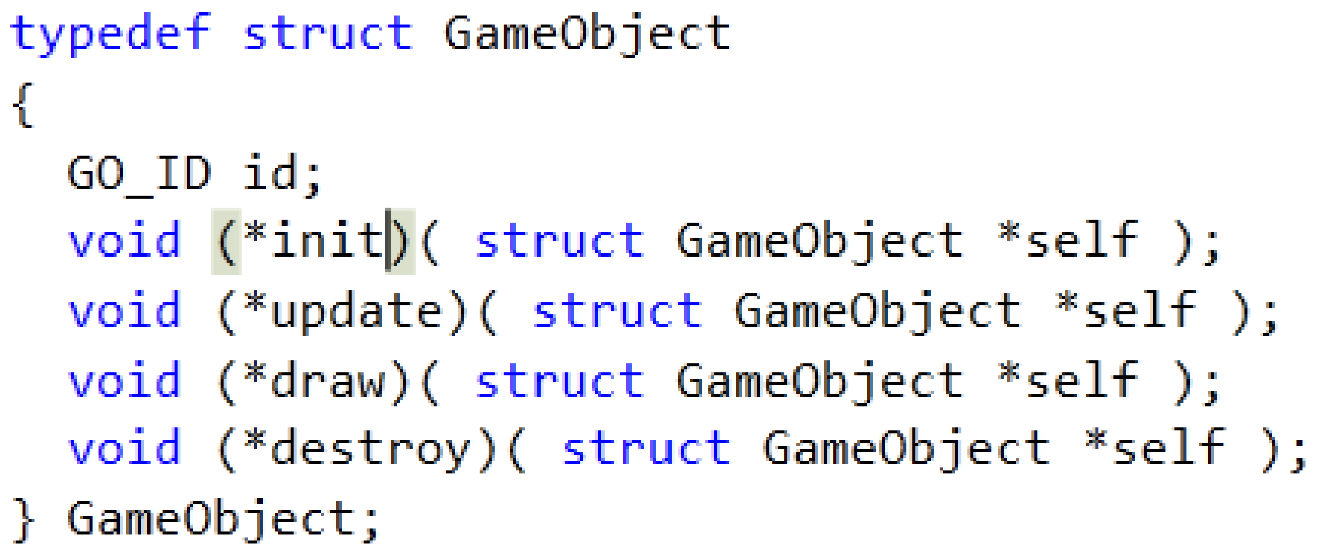


```
typedef enum MESSAGE
{
  M_SETPOSITION,   // Set the position x and y in world
  M_GETPOSITION,   // Get current x and y position
  M_SETVELOCITY,   // Set current velocity of object
  M_GETVELOCITY,   // Get current velocity of object
  M_APPLY_GRAVITY, // Apply gravity to this object
  M_DAMAGE_HP,     // Subtract number from HP
  M_HEAL_HP,       // Add number to HP
} MSG;
```

  - Document each message well!

Messaging

- Send message to GameObject
  - Here’s the old object struct



```
typedef struct GameObject
{
  GO_ID id;
  void (*init)( struct GameObject *self );
  void (*update)( struct GameObject *self );
  void (*draw)( struct GameObject *self );
  void (*destroy)( struct GameObject *self );
} GameObject;
```

Messaging

- Send message to GameObject
  - Here’s the new object struct

Messaging

- Send message to GameObject
  - Here’s the new object struct

```
typedef struct GameObject
{
  GO_ID id;
  void (*init)     ( struct GameObject *self );
  void (*update)   ( struct GameObject *self, float dt );
  void (*draw)     ( struct GameObject *self );
  void (*destroy)  ( struct GameObject *self );
  void (*send_msg)( struct GameObject *self, MSG m ); // <---
} GameObject;
```

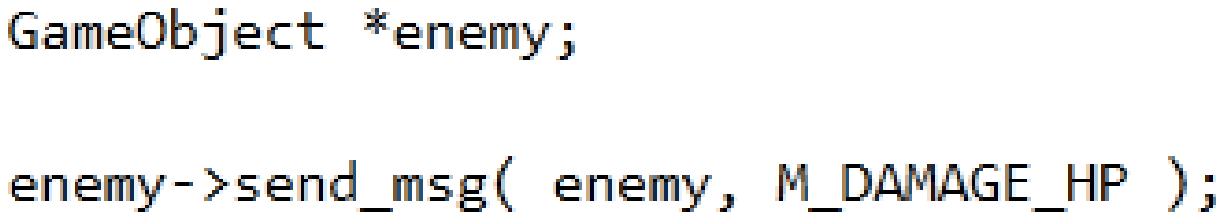
Messaging

- Send message to GameObject
  - Here’s the new object struct
    - Just a function call in GameObject base

```
typedef struct GameObject
{
  GO_ID id;
  void (*init)     ( struct GameObject *self );
  void (*update)   ( struct GameObject *self, float dt );
  void (*draw)     ( struct GameObject *self );
  void (*destroy)  ( struct GameObject *self );
  void (*send_msg)( struct GameObject *self, MSG m ); // <---
} GameObject;
```

Messaging

- Sending a message to an object



```
GameObject *enemy;

enemy->send_msg( enemy, M_DAMAGE_HP );
```

Messaging

- Sending a message to an object

```
GameObject *enemy;

enemy->send_msg( enemy, M_DAMAGE_HP );
```

- Compared to:
  - #include “file.h”
  - Typecasting
  - Searching for the two above!!!
    - Bad…

Messaging

- SendMSG function
  - What is inside?

Messaging

- SendMSG function
  - Switch statement
    - Aka message procedure (proc for short)

Messaging

- SendMSG function
  - Switch statement
    - Aka message procedure (proc for short)
  - Typecast from base to derived here

```
void RedEnemySendMSG( GameObject *self, MSG m )
{
  switch(m)
  {
  case M_DAMAGE_HP:
    RedEnemyDamage( CAST( self, RedEnemy ) );
    break;
  case M_HEAL_HP:
    RedEnemyHeal( CAST( self, RedEnemy ) );
    break;
  case M_TOGGLE_FOLLOW:
    RedEnemyToggleFollow( CAST( self, RedEnemy ) );
    break;
  }
}
```

Messaging

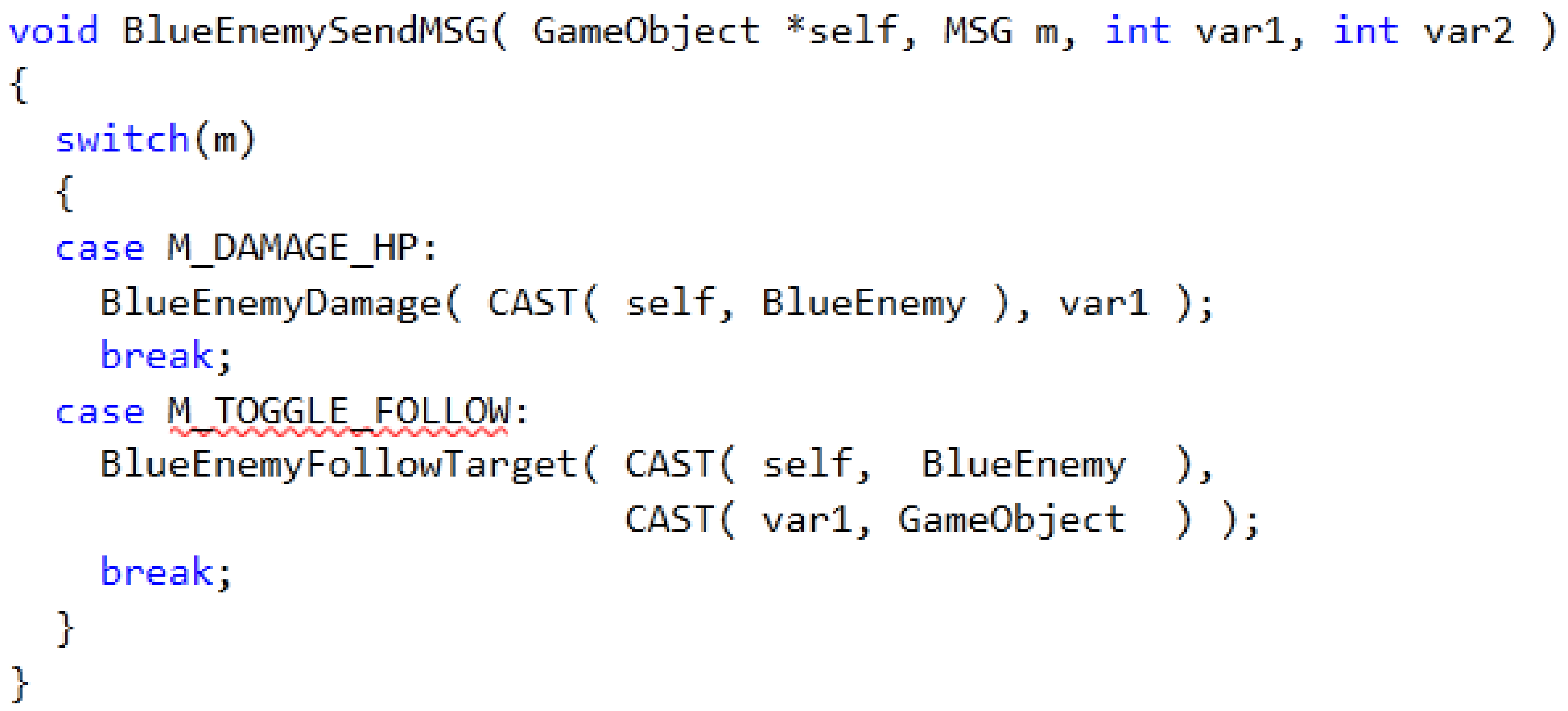
- Awesome! We can:
  - Call functions on inherited
  - Avoid file inclusion
  - Minimal typecasting
- But wait
  - Maybe we need to pass parameters
    - SetVelocity?
      - Need x and y
  - How can this be done?
  - Various types?

Messaging

- I recommend:
  - Two general purpose parameters
    - Integers
- Integers are 4 bytes
  - Pointer is 4 bytes
  - Integer can be cast to any pointer
  - 2 ints = 8 bytes of data to pass
- More than 2 integers
  - Unnecessary
  - Starts getting hectic
  - Pass pointer to struct instead

Messaging

- Two general purpose ints



```
void BlueEnemySendMSG( GameObject *self, MSG m, int var1, int var2 )
{
  switch(m)
  {
  case M_DAMAGE_HP:
    BlueEnemyDamage( CAST( self, BlueEnemy ), var1 );
    break;
  case M_TOGGLE_FOLLOW:
    BlueEnemyFollowTarget( CAST( self,  BlueEnemy  ),
                           CAST( var1, GameObject  ) );
    break;
  }
}
```

Messaging

- Sending a message to an object
  - This time with general purpose parameters
  - Just pass in zero if second param unused

```
GameObject *enemy;

enemy->send_msg( enemy, M_DAMAGE_HP, 10, 0 );
```

- Send address though message:

```
GameObject *Player;
GameObject *enemy;

enemy->send_msg( enemy, M_FOLLOW, (int)Player, 0 );
```

Final Tips

- Passing float through int params
  - Cannot just assign to int
  - Cast to pointer to float instead

```
float x = 1.1f;
enemy->send_msg( enemy, M_DAMAGE_HP, (int)&x, 0 );
```

- #define UNUSED 0

```
#define UNUSED 0

enemy->send_msg( enemy, M_FOLLOW, (int)&Player, UNUSED );
```

Final Tips

- Ask Doug Schilling for advice! He’s awesome
- Keep things as simple as you can
  - Over-complexity is a sign of bad design
  - Are our messages simple?
    - Function call
    - Enum ID
      - Switch statement on ID
    - Two ints
      - Typecast integers if needed
      - Probably most complex
- Ask upper classmen questions
  - Email me: r.gaul@digipen.edu
- Document each message well!

Resources:

- Google wndproc
  - Or Windows Prodecure
    - Windows programming same type of messaging
      - You'll be doing this Sophomore year
      - We wrote our own windowing code in 150
      - CS230 covered basic windows programming
- Refer to Game Object Design ppt
- Game Programming Gems 4: A System for Managing Game Entities
- AsciiEngine (link front page)
  - Implemented this exact type of messaging
- Sample engine on Moodle

Questions

- Anybody have ‘em?