# COMP 1010- Summer 2015 (A01)

Jim (James) Young

young@cs.umanitoba.ca

jimyoung.ca

# Stand alone functions, or, those that change globals

**confusing**

# Stand alone functions

all data comes in through parameters

all data goes out

completely stand alone

only touch globals to read constants

do not modify globals

makes the function easy to use: you don't have to worry about **side effects**.

If you run the function, what else changes?

# Functions that **change globals**

can grab data from globals, not parameters

can modify globals that are not specified by the return

when someone runs it, it can have **side effects**

makes the function hard to use since you need to know what all the impacts will be

use these to break your big code into smaller chunks, but they all work together

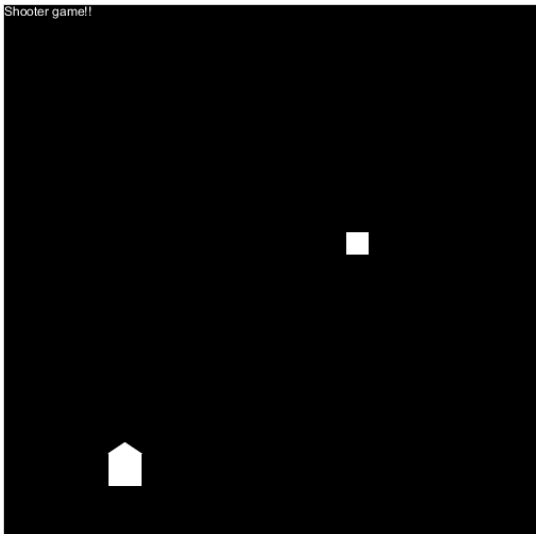
# Divide and conquer

# divide and conquer

**Divide and Conquer**: a great way to break up a complex programming problem into smaller steps

as your programs get longer and longer, this provides a method to enable you to focus on one piece at a time.

# Example problem: spaceship shooter




The space ship is linked to the mouseX.

If the mouse button is pressed, the ship fires a bullet up.

If the user clicks while a bullet is flying, nothing happens.

If a bullet hits the bad guy, game over

Divide and conquer:

- think about the problem in English first.
- turn these steps into new commands
- focus on one element at a time

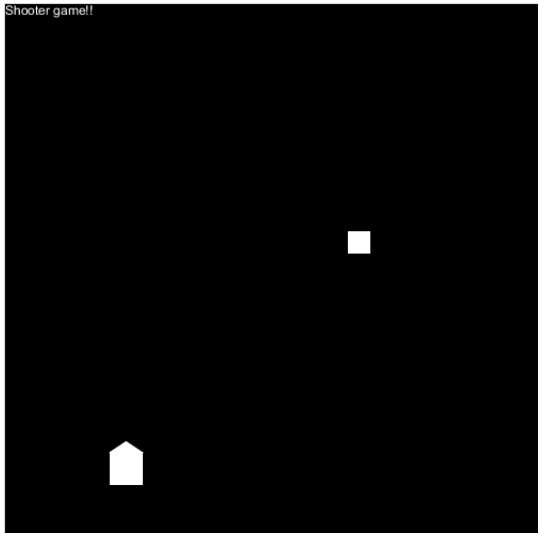
# Let's implement our sample problem

step 1: write the program as a series of steps in comments, in English

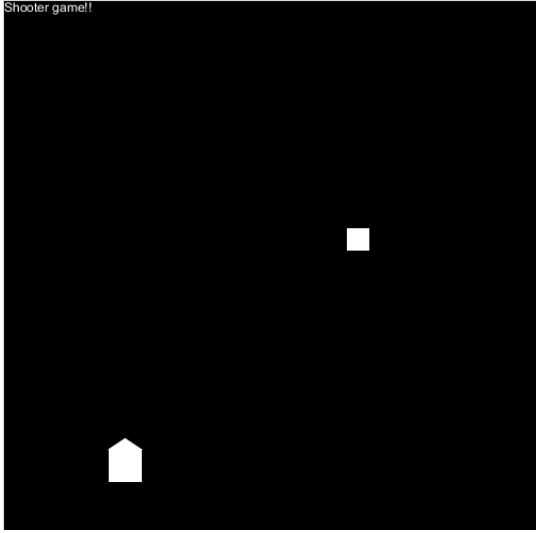
2: turn each step into a function name (command)

3: create the empty functions

4: start implementing the functions



# Steps…




- Draw a title at the top of the screen
- Move and Draw the spaceship
- Check If the spaceship should shoot, and if so, start a bullet
- Move and Draw the Bullet
- Move and Draw the bad guy
- Check if the bullet hit the bad guy, and if so, stop the game
- Draw “WIN” if the game is over.

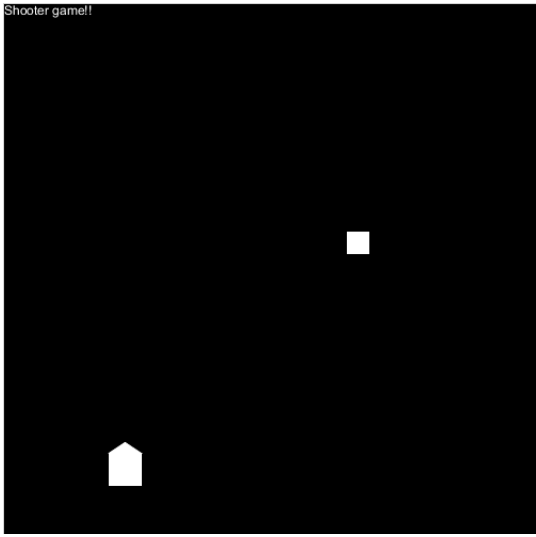
# Let’s implement our sample problem

step 1: write the program as a series of steps in comments, in English

2: turn each step into a function name (command)

3: create the empty functions

4: start implementing the functions



# Turn these steps into commands

```
//Draw a title at the top of the screen
drawTitle();

//Move and Draw the spaceship
 moveAndDrawShip();

//Check If the spaceship should shoot, and if so, start a bullet
checkAndDoShoot();

//Move and Draw the Bullet
moveAndDrawBullet();

//Move and Draw the bad guy
moveAndDrawBadGuy();

//Check if the bullet hit the bad guy, and if so, stop the game
checkBulletHit();

//Draw “WIN” if the game is over.
drawWinMessage();
```

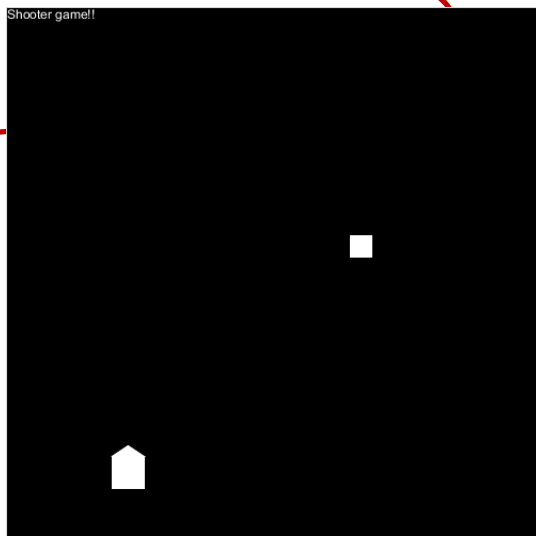
# Let’s implement our sample problem

step 1: write the program as a series of steps in comments, in English

2: turn each step into a function name (command)

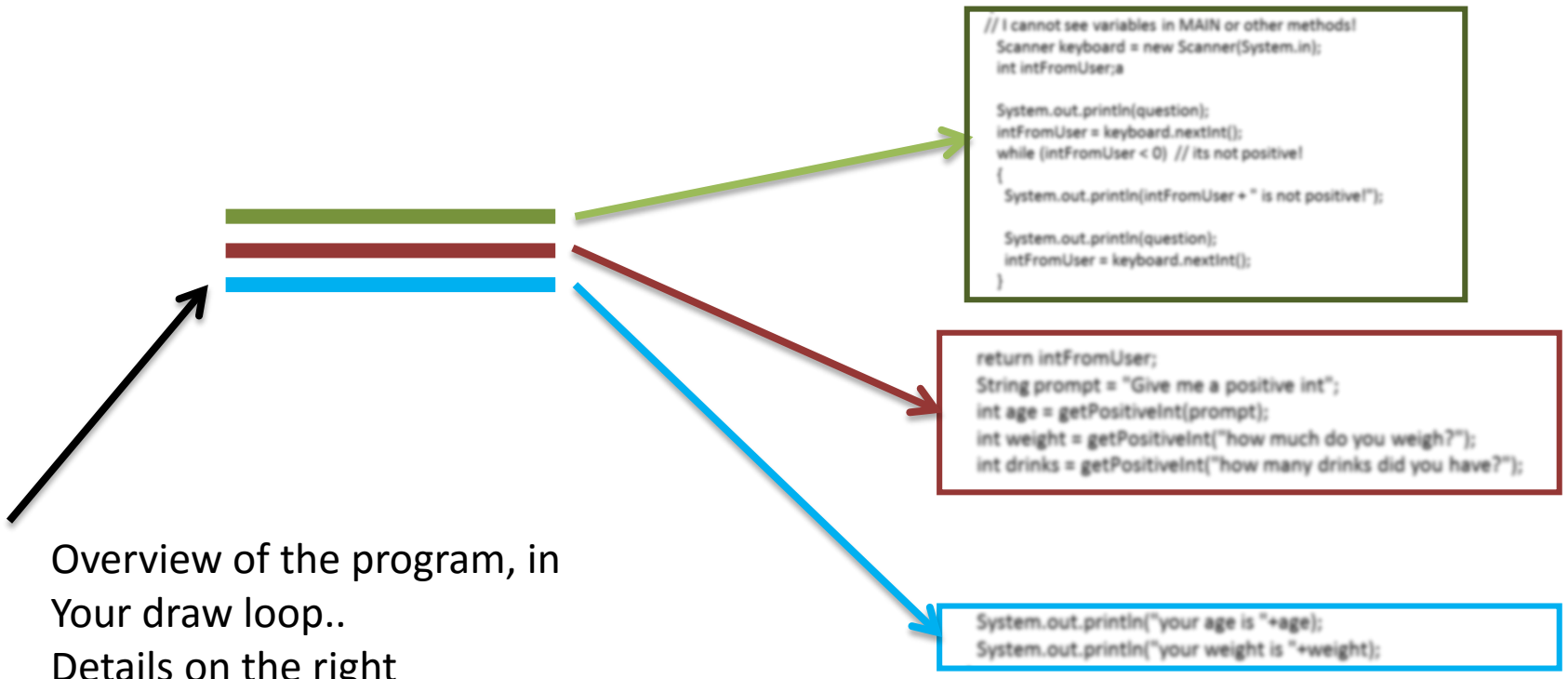
3: create the empty functions

4: start implementing the functions



# Assuming these commands work, everything makes sense

Think of the program as a high-level flow




Overview of the program, in
Your draw loop..
Details on the right

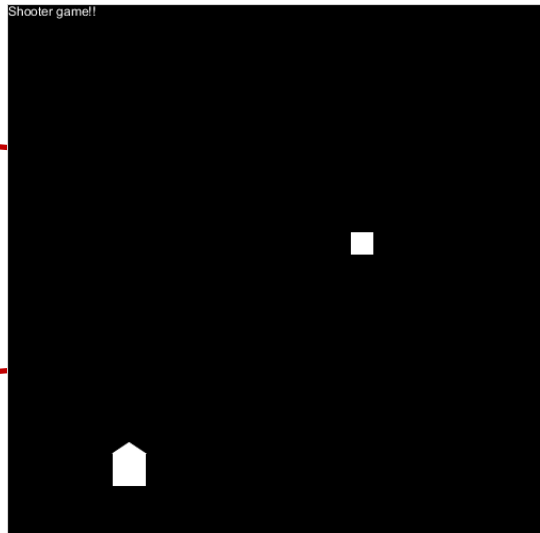
# Let’s implement our sample problem

step 1: write the program as a series of steps in comments, in English

2: turn each step into a function name (command)

3: create the empty functions

4: start implementing the functions



# divide and conquer problem…

dice game:

make an automatic gambling game simulator

- throw six sided dice
- update balance based on roll
    - 1 – you 1.1x your money
    - 2 – you get 1.2x your money
    - 3 – you get 1.3x your money
    - 4 – you get 1.4x your money
    - 5, 6 – you lose 35% of your money

start with $10 and stop when you either reach $1000 or drop below $2

# another example (at home, use divide and conquer! TOUGH)

take a string, and reverse every word in the string, but keep the word order.

“this is a string”

“siht si a gnirts”

# user-defined functions!!

save time by writing something once, and using it again and again!

clean up code by making smaller, easy-to-understand chunks

makes code easier to read and understand!

it gives a name to your operations