Protect Your Base

The “Protect Your Base” project is based off of the old Atari game, “Warlords,” that involves with a ball and 4 bases. The point of the game is to send the ball to your opponent’s base using your defensive paddles. If the base is destroyed, all the blocks defending the base and the paddles disappear. The game ends when a player loses both bases.

Controls

There are two different control scheme based on which mode you are in.

Menu

The game begins with the menu. In here, the controls are completely operated by the mouse (or other similar, pointer-manipulating device). There are four places one can click on:

1) The base formation area (including the middle).
2) Player 1’s color block.
3) Player 2’s color block.
4) Start button.

The base formation area will cycle the base formation to horizontal, vertical, or diagonal base formation. Clicking with the left button will cycle through the formation starting with vertical, then diagonal, and finally horizontal before returning to vertical. The right button cycles through the formation in the opposite order. Similarly, left-clicking the Player 1 and 2’s color block cycles through the colors in this order: yellow, orange, blue, red, violet, and green. Likewise, the right button cycles the color in the opposite order. Finally, left-clicking the start button the game to go to game mode. Incidentally, right-clicking the start button closes the game. This is a feature, not a bug.

Game

The game mode displays 4 bases located in each corner. Each base is protected by 3 layers of defensive (but destructible) blocks: 6 on two sides, and one on the corner. Furthermore, each base has a pair of indestructible paddles, used to deflect the moving ball.

In this mode, the controls changes to the keyboard. Player 1 can move his vertical paddles up using the “W” key, and down using the “S” keys. Similarly, “A” moves the horizontal paddles left; and “D”, right (this is a standard for many computer games). Likewise, Player 2 moves the vertical paddle up with “P” and down with “;”. The horizontal paddles are controlled with “L” being left and “ ’ ” as right.

When one of the player loses both of his/her bases, the game changes back to the menu mode.

Preparation and Requirements

We prepared the project first using Subversion in Google Code. This allowed us to update the code efficiently. Furthermore, we used several different methods to communicate with each other, such as email, instant messaging, and physically.

We found that the Linux operating system came prepared with both OpenGL and a C++ compiler. As such we programmed primarily in Linux.

Progress

The program actually began with little documentation. As such, the game was coded in a Waterfall fashion. Several key features were considered—and successfully implement—throughout the process:

1) The ball bounces in varying degrees.

2) The ball velocity increases over time.

3) Customizable base color.

4) Customizable base formation.

Several problems we bumped into include:

1) Supporting multiple key presses.

2) Menu options changed rapidly when one clicks on it.

3) The ball bounces off of the blocks unrealistically.

4) Changing the ball’s bounce degree, while retaining a consistent velocity.

The methods we used to solve each problem will be discussed in the following section.

Design Decision

The programmer (who is a fan of object oriented programming) used C++’s inheritance and polymorphism features extensively to code the program. Almost the entire program roots from a single abstract class: Graphic. The programmer found that most classes can be expressed with the same traits, mainly:

1) The draw function

2) The animation function

3) X index

4) Y index

Some seemingly common traits turned out to be not-so-useful. This is the case with block traits height and width. While most elements have this feature, ball doesn't.

Programming the basics

The code was built up in a gradual matter, first beginning with the window and the ball. Initially, the program began with a single ball moving at a random direction, bouncing off of the window borders. The Ball class drew the ball, as well as defined the bounce algorithm. The window, on the other hand, is merely defined by the main file.

It's also worth noting that the problem about retaining the ball's velocity was neither fixed nor considered. It was decided that to keep the program efficient, the solution had to be dropped.

The Block class was defined next, used to demonstrate the ball bounce on an independent object. As presented before, we revised the collision physics 4 times until it worked realistically.

This class was further extended with DestructibleBlock and Paddle. DestructibleBlock stopped drawing itself after it came in contact with the ball. Paddle defined movement in a vertical or horizontal fashion.

Putting colors and keyboard controls

From here, we began developing classes that contained Ball, DestructibleBlock, and Paddle. The Base class was created to help setup the DestructibleBlock and Paddle objects to form each corner. When we programmed the Base class, though, it became apparent that we had to create 2 more classes: one to define the keyboard controls, and one to define the color.

The Control class was created when we realized that multiple keys were not initially supported. The only way to implement this feature was to hold onto an array of Booleans, indicating which key was held down, and which one wasn't. To minimize copying the control code, we created a class to be used in many future classes.

The Color class was created with a similar intent. We found that there wasn't that many colors used in the game. As such, we created Color to reduce repetition.

Using the basics

Base class contains a Control, four Colors, 14 DestructibleBlocks, and two Paddles to generate a corner base. The Base's purpose is to use these elements to a comprehensible manner. The Base sets up:

1) The dimensions for each Block-extending class.

2) Assign the Colors to each block.

3) Assign each DestructibleBlock with different roles (one is the Achilles' heel, the rest are defensive).

4) Correlate the Control to the Paddle movements.

Each corner had a different DestructibleBlock and Paddle coordinates, so we created four private functions to generate each one.

Rather than immediately creating a Game class, we instead created a Player class: a class that contains two Bases. Since the Base color and control configuration are limited, we used a new class to enforce this.

Finally, the Game class was created to store and compute the complete game mode. It holds two Players and one Ball. The Game class enforced further limitation on Player, as well as compute Ball positions based on the computation from the window dimension and the Block dimensions.

Defining global values
A little sidestep from the classes: with the creation of the Game class, it became apparent that several constant variables should be defined globally for all classes. Thus, the global.h file was created, defining the window dimensions. A few macros were also defined for both debugging and deleting pointers a little more safely.

Menu and mouse
The next step was to create the menu. First, the mouse class had to be created. Using the GLUT mouse function template, we created a class that updates using those very variables. The significant function in Mouse was the get_button, which only returned a value if the button was in the pressed-down state.

The Menu class computed many things that could have been separated to specialized classes. However, time was tight. The Menu class contains the mouse controls, game setting variables, and several Block objects. Whenever the Block object (which also detects mouse-hover) recognizes the mouse press, the setting variables are updated respectively.

Putting it all together
The ProtectYourBase class puts Menu class and Game class together to create the final product. This class held the Color objects, Mouse object, and Control pointers shared in Menu and Game. When the Menu changes its Color, it updates the ProtectYourBase and Menu Color at the same time.

The main.cpp, finally, uses the ProtectYourBase to create the graphic, animations, and controls. In addition, the main.cpp delays the mouse controls if the Mouse is held down, using various Boolean functions on the animation function.

Note that test_menu.cpp and test_game.cpp were created to test Menu and Game class respectively. They serve no other purpose.

Code

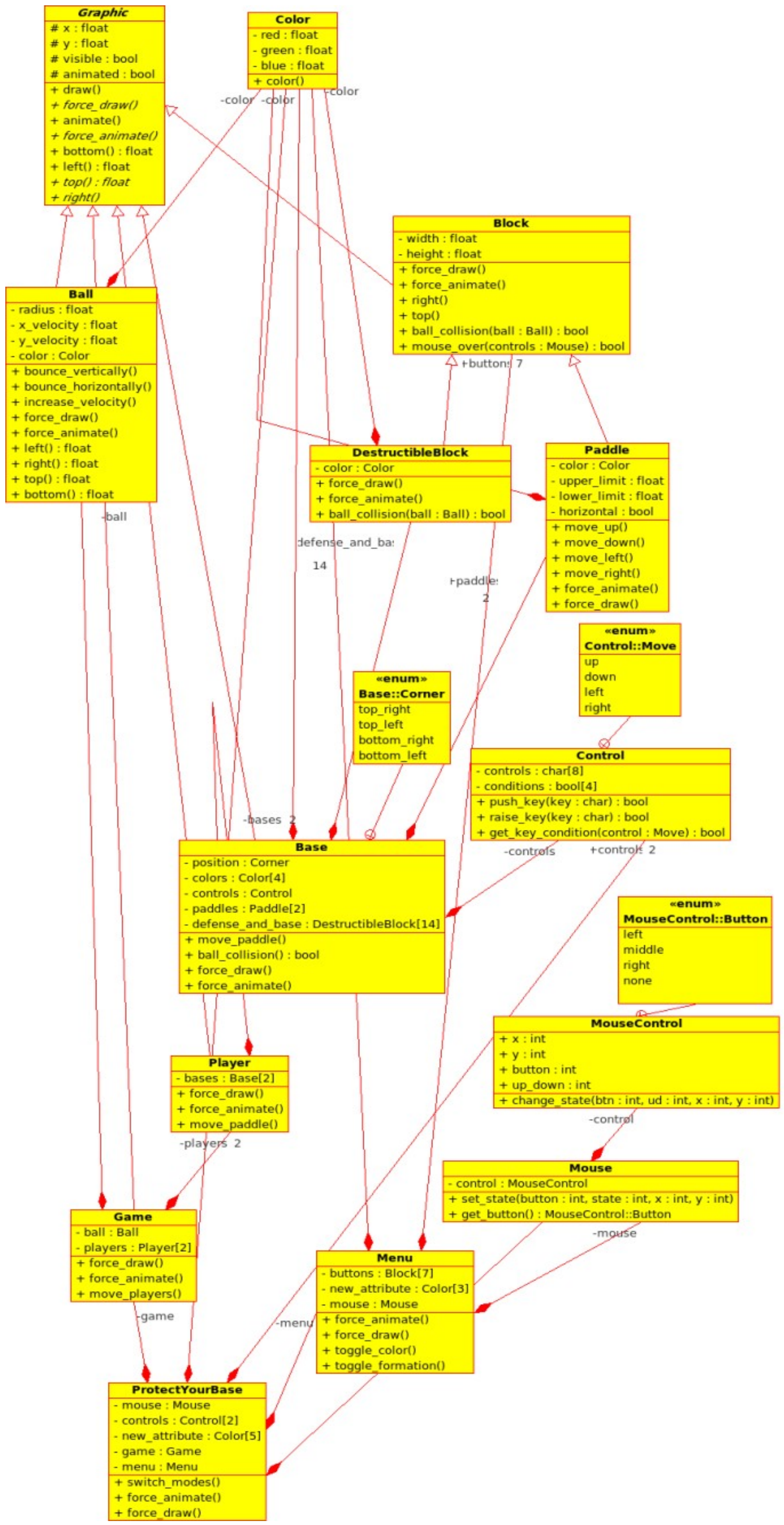
Code is available at http://code.google.com/p/protectyourbase/source/browse/#svn/trunk. This can be downloaded using subversion. Directions are available at http://code.google.com/p/protectyourbase/source/checkout.

Compile the code by calling “make”. This will create 3 executables: “menu”, which tests the Menu class; “game”, for the Game class; and “ProtectYourBase”, the complete package.

There is a known problem. The mouse control becomes dysfunctional after resizing the window size. As such, avoid resizing the window as much as possible.

Finally, there is a class that never came to use. The Image class was intended to replace the Color objects in Block. However, we never found time to implement this, and the class remained.

Class diagram
Enjoy->



Graphic
# x : float
# y : float
# visible : bool
# animated : bool
+ draw()
+ force_draw()
+ animate()
+ force_animate()
+ bottom() : float
+ left() : float
+ top() : float
+ right()
Color
- red : float
- green : float
- blue : float
+ color()
-color
-color
-color
Block
- width : float
- height : float
+ force_draw()
+ force_animate()
+ right()
+ top()
+ ball_collision(ball : Ball) : bool
+ mouse_over(controls : Mouse) : bool
+buttons 7
Ball
- radius : float
- x_velocity : float
- y_velocity : float
- color : Color
+ bounce_vertically()
+ bounce_horizontally()
+ increase_velocity()
+ force_draw()
+ force_animate()
+ left() : float
+ right() : float
+ top() : float
+ bottom() : float
-ball
DestructibleBlock
- color : Color
+ force_draw()
+ force_animate()
+ ball_collision(ball : Ball) : bool
Paddle
- color : Color
- upper_limit : float
- lower_limit : float
- horizontal : bool
+ move_up()
+ move_down()
+ move_left()
+ move_right()
+ force_animate()
+ force_draw()
defense_and_ba
14
+paddle
2
«enum»
Control::Move
up
down
left
right
«enum»
Base::Corner
top_right
top_left
bottom_right
bottom_left
Control
- controls : char[8]
- conditions : bool[4]
+ push_key(key : char) : bool
+ raise_key(key : char) : bool
+ get_key_condition(control : Move) : bool
-controls
+controls 2
-bases 2
Base
- position : Corner
- colors : Color[4]
- controls : Control
- paddles : Paddle[2]
- defense_and_base : DestructibleBlock[14]
+ move_paddle()
+ ball_collision() : bool
+ force_draw()
+ force_animate()
«enum»
MouseControl::Button
left
middle
right
none
MouseControl
+ x : int
+ y : int
+ button : int
+ up_down : int
+ change_state(btn : int, ud : int, x : int, y : int)
-control
Player
- bases : Base[2]
+ force_draw()
+ force_animate()
+ move_paddle()
-players 2
Mouse
- control : MouseControl
+ set_state(button : int, state : int, x : int, y : int)
+ get_button() : MouseControl::Button
-mouse
Game
- ball : Ball
- players : Player[2]
+ force_draw()
+ force_animate()
+ move_players()
-game
Menu
- buttons : Block[7]
- new_attribute : Color[3]
- mouse : Mouse
+ force_animate()
+ force_draw()
+ toggle_color()
+ toggle_formation()
-menu
ProtectYourBase
- mouse : Mouse
- controls : Control[2]
- new_attribute : Color[5]
- game : Game
- menu : Menu
+ switch_modes()
+ force_animate()
+ force_draw()