

Angry Monkeys

In the game of Angry Birds, the player sling-shots some rather surly birds of various sizes, shapes, and colors at a structure of rocks, wood, glass, and pigs. Points are gained by damaging the structure and destroying pigs.

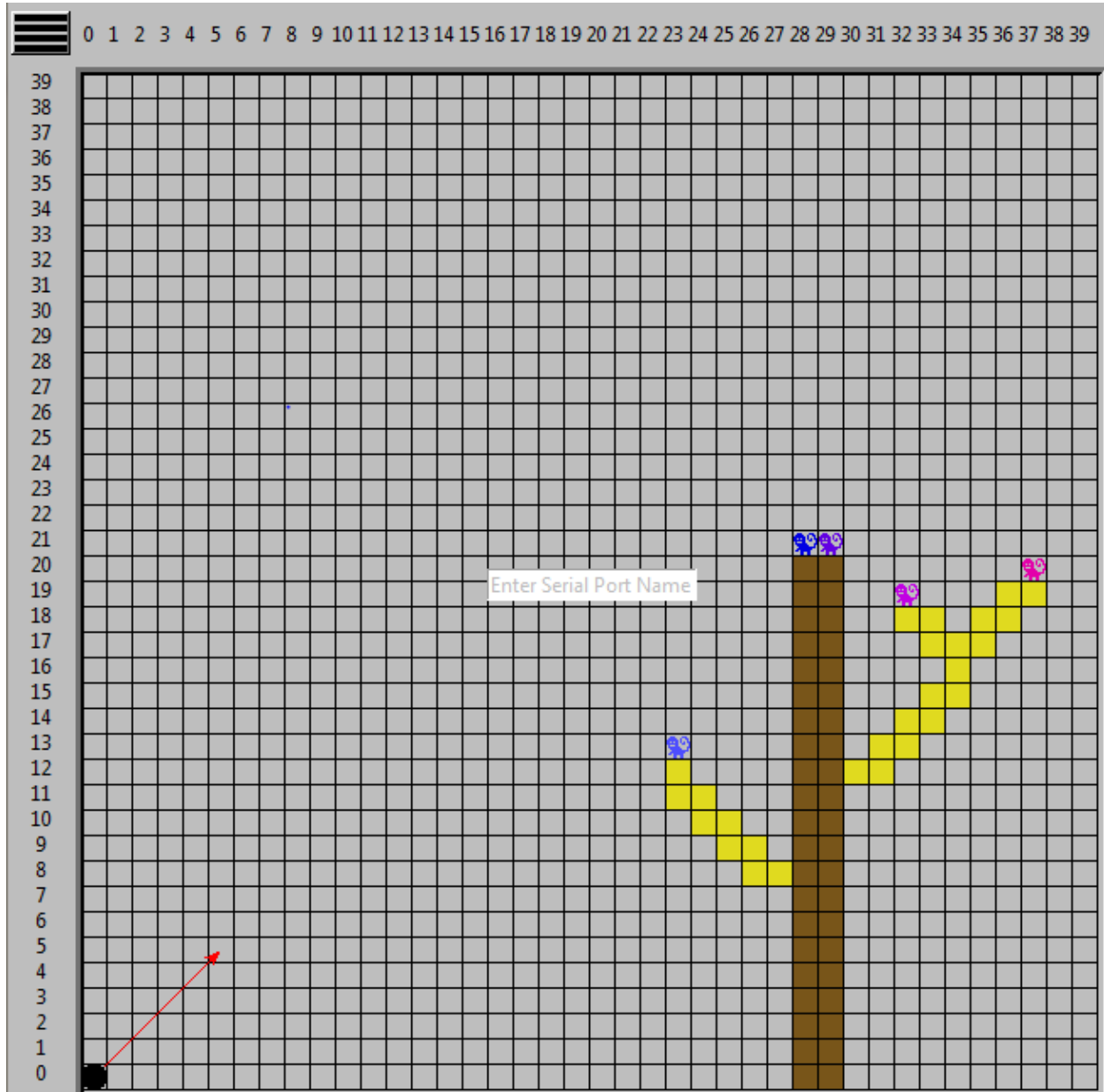


Figure 1. A typical Angry Monkeys game world.

The goal of this project is to program a much simpler (and less copyright infringing) game in the genre of Angry Birds that we'll call Angry Monkeys. You will start with a socket communication-based software framework that provides all of the necessary tools you need to play the game. The game will establish a communication link between a front-end python program that displays a graphical user interface and a back-end C program that accepts user keystrokes and computes the animation effects as well as providing hints to the user. The front-end program will draw the game world and send a copy of the world to the back-end. Then, you

will develop the game engine within the back-end and compute the front-end GUI commands and hints to allow a user to play the game.

Implement Angry Monkeys

First, download and unzip **P2_sock.zip** from the class Assignments web page. Follow the instructions in the **readme-socket.txt** file to set up the needed software socket-based framework.

Next, you will develop the game engine and control interface needed to play a game of Angry Monkeys. The front end will load a game world and send a sparse array representation of the world to your program. Your program will need to do three things: use the user keystrokes to take in user inputs, calculate the game physics and collisions based on user inputs and the game state, and provide hints for how to hit the monkeys.

A number of functions that will be needed to send messages to the front end are provided in the main.cpp shell program (in P2_sock). A description of their usage and functionality can be found in '*Messaging Functions.pdf*' in the project files.

The zipped P2_sock directory also contains two demo file executables (**demo32** for 32-bit systems and **demo64** for 64-bit systems). These demonstrate part of the game animation and illustrate proper functioning of the socket communication.

Part A: User inputs

The first step is to retrieve user inputs via keystrokes to determine a shot trajectory. You should use two keys (C and X) to move the shot angle up and down, one key (V) to toggle the shot power between high and low, and one key (Z) to trigger a shot.











The angle should be limited to 0-90 degrees, and the power should toggle between the high and low values defined by PHIGH and PLOW in the provided shell code. The angle and power inputs should be sent to the front end using the paaUpdate(power, angle) function.

Part B: Game Engine

The next step is to create the game engine that calculates the animations and collision detection for the cannon shots based on user inputs and game world state. The guidelines for your game engine are as follows:

1. Game World Array:

An example game world can be seen in Figure 1. The cannonball and shot indicator are not part of the world array. **The game world is made up of a tree, which is comprised of tree and branch tiles, and monkeys. Each tile has a strength, the number of shots it can sustain before it is destroyed. The color of each tile/strength combination is shown below:**

Type/Strength	1	2	3	4	5
Tree					
Branch					



When the game starts, **the state of the game world is given to you as a sparse array** in a format similar to the sparse vectors in Homework 2. The array will be in the following format:

```
int world[N];
where
```

world[0]	Total size of array (#rows * #columns)
world[1]	Number of non-empty squares
world[i*4+2] for i = 0, 1, 2, ... world[1]-1	Row index of i th non-empty square
world[i*4+3] for i = 0, 1, 2, ... world[1]-1	Column index of i th non-empty square
world[i*4+4] for i = 0, 1, 2, ... world[1]-1	Type of object in i th non-empty square
world[i*4+5] for i = 0, 1, 2, ... world[1]-1	Strength of the i th non-empty square

The type field will be an ASCII character code corresponding to 'M', 'T', or 'B' for monkeys, tree trunks, and branches respectively. 'M', 'T', and 'B' are ASCII codes 77, 84, and 66. The strength field will be an integer 1-5. **It can be assumed that the game world is always square.**

As an example, take the sparse array:

```
{9, 3, 1, 1, 77, 3, 2, 0, 66, 1, 2, 2, 84, 5}
```

This would represent a 3x3 array with 9 total squares. 3 of the 9 squares are non-empty. The first non-empty tile is a monkey with strength 3 at row 1, column 1. The second non-empty tile is a branch with strength 1 at row 2, column 0. The final non-empty tile is a tree with strength 5 at row 2, column 2.

2. Shot Physics:

Shot trajectories should be calculated using the kinematic equations for projectiles. Gravity will be defined in the shell code as GRAVITY. The time-based kinematic equations for a projectile initially located at x=0, y=0 are as follows:

$$y(t) = v_y \cdot t - \frac{1}{2} \text{GRAVITY} \cdot t^2$$

$$x(t) = v_x \cdot t$$

It may be useful to rearrange these by substituting $t = \frac{x(t)}{v_x}$ into $y(t)$ to give:

$$y(x) = \frac{v_y}{v_x}x - \frac{1}{2}GRAVITY \cdot \left(\frac{x}{v_x}\right)^2$$

Additionally, v_x and v_y are given by:

$$v_x = POWER \cdot \cos(ANGLE)$$

$$v_y = POWER \cdot \sin(ANGLE)$$

Where POWER and ANGLE are determined by user inputs. Sine and cosine functions can be found in the <math.h> standard C library.

IMPORTANT: To ensure your x and y values are consistent with the auto-grader and compatible with the discrete nature of the game world, you should use the floor() function from <math.h> to truncate your x and y values i.e. use the following equations in your code:

$$\begin{aligned} y(t) &= \text{floor}(v_y \cdot t - \frac{1}{2}GRAVITY \cdot t^2) \\ x(t) &= \text{floor}(v_x \cdot t) \\ y(x) &= \text{floor}\left(\frac{v_y}{v_x}x - \frac{1}{2}GRAVITY \cdot \left(\frac{x}{v_x}\right)^2\right) \end{aligned}$$

3. Collisions:

3A Basic Collisions:

When you calculate shot trajectories, your cannon balls should stop when they hit something, i.e. when they reach a tile occupied by a tree, branch, or monkey. When a tile is hit, you should decrement the tile's strength. When a tile's strength reaches 0, it should be deleted.

If your program correctly detects collisions, updates tile strength, and deletes tiles properly, this is worth 15 points.

3B Advanced Collisions:

To receive more points, the effects of destroying a tile should be propagated to nearby tiles in the following ways:

- Monkeys:** When a monkey is deleted, it does not affect its surroundings; only the single monkey should be deleted.
- Branches:** When a branch tile is destroyed, any branch tiles that were supported by that tile and any monkeys sitting on the branch should fall. When a branch is deleted you will need to find all branch tiles that are no longer attached to a tree directly or through other branch tiles and delete them. You should also find any monkeys that were supported by those branch tiles and delete them as well.
- Trees:** When a tree tile is destroyed, the tree tiles directly above it should all be destroyed and any branches attached to it should be destroyed according to

(b.). Any monkeys supported by the tree and branches deleted in this way should be deleted as well.

The '*Collision Examples.pdf*' file included in the project files shows some before and after pictures of collisions that you can use as example cases.

If your program implements the advanced collision guidelines and properly propagates tile deletion, this is worth 15 points.

4. Animation:

The ultimate function of a game engine is to examine user inputs, calculate how those inputs affect the game state, and then drive the proper outputs. In this case, the proper outputs are animation effects.

To send animation updates to the front-end program, you will need to use the messaging functions found in '*Messaging Functions.pdf*'. To draw the cannonball in flight, the `updateShot(row, col, delete)` function can be used to draw a new cannonball at (row, col); the delete parameter determines whether other cannonballs on screen should be deleted before drawing the new ball. The delete parameter can be used for debugging but should be set to 1 (True) in your final submission. **You should update the cannonball's position at least 4 times as it is flying;** more updates will make the shot trajectory look better. When a shot hits a tile, the `colorTile(row, col, strength)` function can be used to update a tile's strength (and therefore color) and the `deleteTile(row, col)` function can be used to delete the object at (row, col). Tiles should be updated and deleted according to the collision guidelines in (3.). **Tiles can be recolored and/or deleted all at once when a hit occurs, or you can add your own creative timing/animations as long as the final state follows the collision rules.** Finally, as a user is selecting shot parameters with keystrokes, you should send the power and angle changes to the front-end using the `paaUpdate(power, angle)` function.

If your program signals correct animations, this is worth 15 points.

Part C: Game Hints

The final step of this project is to calculate a hint for what shot the user should take. When the game starts and your program receives the world, before the game engine starts running, your program should first find the location of the leftmost monkey in the game world and calculate a power and angle combination that will destroy that monkey.

You can choose to either:

- a. **Provide a hint that hits the left-most monkey directly. Correctly providing this hint is worth 30 out of 45 possible points for hint accuracy. Or:**
- b. **Provide a hint that destroys the branch on which the left-most monkey sits. Your hint should target the branch tile that destroys the entire branch, the branch tile that forms the joint between branch and tree. Correctly providing this hint is worth 45 out of 45 possible points for hint accuracy.**

Your hints do not need to account for another object being in the way of the target i.e. as long as the shot you specify would pass through the location that you specify if the world was totally empty, it is considered correct. *The location you specify must also correctly point to the left-most monkey or the corresponding branch joint.* To report a hint to the front-end, you should call the `hint(row, col, power, angle)` function with the proper shot parameters and location.

Comments on calculating hints:

As you may realize, there are many combinations of power and angle that would send a projectile through a give location. To help bound the problem, the possible shot power is limited to one of two values and the angle is limited to 0-90 degrees. However, even with these restrictions, there are multiple correct answers.

Additionally, the closed form solution for calculating a hint is rather complex. The equation of a shot curve is shown again below:

$$y = \frac{v_y}{v_x} x - \frac{1}{2} G R A V I T Y \cdot \left(\frac{x}{v_x} \right)^2$$

Substituting v_x and v_y with power and angle terms and using the monkey location for x and y :

$$y_{monkey} = \frac{P O W E R \cdot \sin \theta}{P O W E R \cdot \cos \theta} x_{monkey} - \frac{1}{2} G R A V I T Y \cdot \left(\frac{x_{monkey}}{P O W E R \cdot \cos \theta} \right)^2$$

$$y_{monkey} = \tan \theta \cdot x_{monkey} - \frac{1}{2} \frac{G R A V I T Y}{P O W E R^2} \sec^2 \theta \cdot x_{monkey}^2$$

As can be seen, even for a fixed power and location, solving for the required angle involves some rather nasty trigonometry. With this in mind, **you will likely have to come up with an iterative method that finds the right shot trajectory.** You should not simply check every power/angle combination, but rather make an educated guess about a possible power/angle combination and then make iterative changes to the angle until the correct trajectory is obtained. **It is also possible that you could create your own novel algorithm for finding hints.**

Project Submission

For your solution to be properly received and graded, do the following:

Rename your `main.cpp` file to **P2.cpp** and upload it to the submission site before the scheduled due date, **9:00pm on Wednesday, 27 November 2013.** **There will be a one hour grace period, after which no submitted material will be accepted.**

You should design, implement, and test your own code. Any submitted project containing code (other than the initial setup code) not fully created and debugged by the student constitutes academic misconduct.

Project Grading: The project weighting will be determined as follows:

<i>part</i>	<i>description</i>	<i>due date</i>	<i>percent</i>
P2	Angry Monkeys Program (rename main.cpp to P2.cpp and upload)	Wednesday, 27 November 2013 before 9 p.m.	
	Technique and Style		10
	Animations		15
	Basic Collisions		15
	Advanced Collisions		15
	Hint Accuracy: <ul style="list-style-type: none">- Direct monkey hint – 30 points- Branch joint hint – 45 points		45
	<i>total</i>		100

Note that your program will be graded based on newly created worlds in addition to the ones provided in the Project support files.