# CSC171 — Project 1

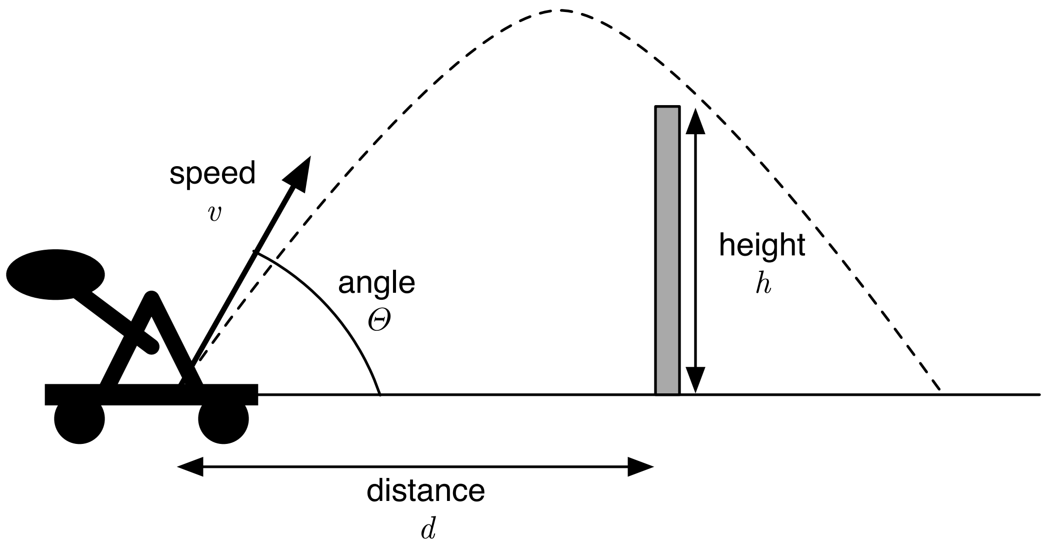
## TTY Projectiles

People have played computer games since before there were computers. For example, Alan Turing, one of the giants of Computer Science, developed a program to play chess using a formal model of a computing device that never existed physically.

Before computers had color monitors and fancy graphics, the only way they could communicate was by printing on a sort of typewriter called a "teletype" or "TTY." Eventually display screens replaced the typewriters, but these were still called TTYs because they could only display text and the user could only interact by typing text. Our Java programs that input using a `Scanner` and output using `System.out.println` can be called "TTY programs" because they also interact only through text. There is a long history of text-based or TTY computer games, for example Colossal Cave (a.k.a. ADVENT) and the Zork series.

In this project you will implement a very simple TTY game. In this game, the user has a catapult that can launch projectiles. In each round of the game, the computer places a wall in front of the user. The user aims their catapult by setting the launch angle and speed. The computer then computes whether the projectile makes it over the wall and informs the user. The user gets points for clearing the wall and loses points for hitting the wall. The game continues through successive rounds until the user quits.

The following graphic illustrates the game. Note however that you will *not* be implementing any graphics in this project. This is a *TTY* game.



## Suggestions

Use the following suggestions to guide your design and implementation:

- You will need to use variables to represent the state of the game for your program. For example, you will need a variable storing the distance and height of the wall for the current round. You will need to keep track of the user's score. And so on. Properly designing this "model" of the game is crucial to developing a good program.

- Your program will consist primarily of the "Print-Read-Compute" loop (or loops) necessary to implement the gameplay described above. Obviously for the loops you will use appropriate iteration statements. For reading input, use a `Scanner`. For interpreting the user's input and deciding what happens in the game, you will need to use conditional statements. These will be nested within the loops, and you will may also need nested conditionals to figure out what to do.

- The game would be pretty boring if the wall was always the same height and at the same distance. So we will need some *randomness* to make things more interesting. You will use the `java.util.Random` class to generate random numbers, and use these to make random choices.

- We will ignore the effects of drag and assume an ideal projectile. You can look up the equations of motion for a projectile, for example Projectile_motion and Trajectory_of_a_projectile at Wikipedia. However the main formula that you need for this project is that for a projectile launched from the ground (height zero) at speed $v$ and angle of elevation $\theta$, the height of the projectile when it is at distance $x$ is:

$$y = x \tan\Theta - \frac{gx^2}{2(v\cos\Theta)^2}$$

  where $g$ is the acceleration due to gravity (approximately 9.8 $\mathrm{m/s^2}$).

## Requirements

- Your program *must* be well-formatted according to standard conventions for Java as seen in the textbook and in class.

- Your program *must* be well-commented to explain what the code is doing. Code with no comments, or code whose comments are not correct will be penalized.

- Your submission *must* be a ZIP archive containing your Java source files and any supporting files.

- Your submission *must* also include a file named "README.txt" describing how to run your program, any design decisions you made that you feel the need to explain, and anything else that you think will help us understand your project.

- Whenever your program wants input from the user, it *must* prompt for it with an appropriate message printed *without* a newline.
- After each launch, your program *must* inform the user how far the projectile was from the top of the wall (above or below) and what the outcome was. Something like the following:
  - Over the wall by a few (like three) meters or less: “you made it”
  - Over the wall by more: “plenty of room”
  - Hit the wall within a few meters of the top: “not quite over”
  - Hit the wall father down: “not even close”

  Of course your program should use whole sentences and try to make it interesting for the user. You should adjust the parameters of the game (minimum and maximum distance and height of wall, distances for the various ranges above) to make the game challenging but playable.
- Your program *must* keep track of the user’s score. Each launch costs 1 point. A close clear gets 5 points (so +4 net). A far clear gets +2 net. A near miss loses 1 point net and a far miss loses 3 points net. Your program must inform the user of their score after each launch and at the end of each round. You may adjust these scoring rules—document any changes in your code and in your README.
- The user *must* be allowed to quit after each round. You may optionally allow them to pass on a round (give up) or quit completely during a round.

## Extra Credit

You can get full points for accomplishing the requirements as described above. The following are some ideas for extra credit, up to a maximum of 20%.

- Adding some additional randomness into the computer’s output is an easy way to make a simple game less boring. For example, you can generate different sentences to tell the user the outcome of their launch. [up to 5%]
- In the real world of course there is drag from wind resistance. Including this in the calculations is tricky, but doable. If you want to do this, please have your program print what the result would be without drag as well as the more realistic results. [up to 10%]
- You could have the wall moving towards the user at some (relatively slow) speed. This affects the distance to the wall even if you assume no motion once the projectile is launched. You would need to use the elapsed time before the first launch and between later launches to get this right. If the target gets to the user before they launch, you can describe what happens. Note that you won’t get a chance to compute this until after

it has happened, since your program will be waiting for the user's input while they are being crushed by the wall. [up to 10%]

- Please do not worry about graphics or other forms of user interfaces for this project. We will get to those soon enough. No extra credit for them in this project.
- Other ideas: document what you did *clearly and prominently* in your documentation. We'll give you credit based on our assessment of how hard it was to do (but the total extra credit can never be more than 20%).

## Collaboration Policy

You will get the most out of this project if you do the work yourself. Watching someone else program is no substitute for doing it yourself. Doing the projects yourself will improve your performance in the rest of the course.

However, collaboration on projects is permitted, subject to the following requirements:

- Groups of no more than 3 students, all currently taking CSC171.
- You must be able to explain anything you or your group submit, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the group should submit on the group's behalf and the grade will be shared with other members of the group. Other group members should submit a short comment naming the other collaborators and which of them submitted the actual project.
- All members of a collaborative group will get the same grade on the project.

Any other submission of code that is not your own constitutes a violation of the University's Academic Honesty policy.