ParSi 1.0

Benjamin Tatman

1 Before you start

1.1 What is ParSi?

ParSi is a Particle Simulator. It is a relatively simple Particle Simulator, and does not attempt any graphics, instead prefering to release the data in either a nice tabulated format, or a CSV file.
It was originally written over the summer, and was designed during a few maths lessons. Enough with the back story.
It can output in two ways and take input in three ways. There is also the option to use radians, if this is applicable to you. This will be explained over the following pages, along with an explanation of how ParSi works, incase you wish to get involved.

1.2 Giving Feedback

ParSi is still in active development, and should you wish to want to get involved, I would recommend reading this document. If you want to, you can fork it on GitHub, at *http://www.github.com/ThatPerson/ParSi*, and if you do could I please request that you send me an email at *ben@tatmans.co.uk* in order to give me some idea of who is working on it, and what you are trying to do. This will hopefully allow me to help you get used to the codebase.
If you are not a programmer, and just want to use ParSi, and it helps you with what you are doing, please leave me a note at the same email address as above! I very much enjoy reading what people are doing with it, and it will also allow you to give suggestions. If you are writing a paper and using ParSi to help, and the paper is not restricted (ie, if it is a paper which has been released to the world and is online) please also send me a link! It is always nice to read about what people have done with it.

2 Arguments

ParSi has a few arguments, all of which can be seen with a simple $ParSi-h$. They are also listed below.

```
-c Enter CSV output mode
-r Use Radians as opposed to degrees
-if Use .sim file as basis for simulation
-of Put output to file.
-i enter interactive mode.
```

These are largely self explanatory, and could be used as follows:

```
ParSi -c -r -if file.sim -of output.csv
```

Which would read in a 'sim' file and output a CSV file called 'output.csv' in radians.

3 Inputs

ParSi can take input in one of three ways. You can either compile it in (this will not be faster, but may allow you to do more advanced things, such as modifying forces in the simulation), use the interactive mode, or write a .sim file.

3.1 Compiling it in

I would not necessarily recommend doing this, however it is possible. When doing this, there are a few functions and structs you should be aware of. These are as follows.

```
42 typedef struct {
43         char name[500];
44         Force speed;
45         Force force;
46         Position pos;
47         int mass;
48         int shown;
49 } Particle;
```

The Particle definition, and can be initiated with Particle p[], or such like. It keeps track of the speed (I know it is a Force definition, however the Force struct is a Vector. It is still named Force as I have not gotten around to

changing it), the forces affecting it, the position (x and y), the mass, whether or not it is shown, and the name of the particle.
Rarely are Particles seen alone, more often are they seen in arrays. This is shown in the code I use lower down, where I use the wait_all function.

3.1.1 wait_all()

wait_all is a function you will likely use a lot. It is defined as:

```
356 void wait_all(Particle p[], int count, float time,
float display_time, int show_headers, int radians,
FILE * output);
```

While this may appear daunting, it is not very much so. It consists of the following:

```
Particle p[];
int count;
float time;
float display_time;
int show_headers;
int radians;
FILE * output;
```

These are quite verbose. 'p' is an array of Particles, specifically the particles in the collision. Due to how the 'Particle' definition is, you are required to pass 'count' as the number of items in the array. 'time' tells it how long to move them for, and 'display_time' tells it what to pass to tabulate_particles().
Then, we have 'show_headers'. This tells tabulate_particles() whether or not it needs to show the headers of the file, and if used in a for loop this should only apply to the first item. If not, it ends up messy. 'radians' tells it whether or not it should use, well, radians, and 'output' is a FILE to output to. You could set this as stdout, or another FILE object. This is how the '-of' parameter works.

3.1.2 balance_force()

Another useful function is balance_force(). This simply takes two forces as arguments, and returns the average, which can be very useful for collisions or gravity.

Warning. Maths ahead. It does this by utilising yet another function, called resolve. Resolve turns a force into a x and a y force, by using the knowledge that if the angle is between certain parameters, then the direction of the force will be in a certain direction.
Ie, if the angle of the force is 23°, then you know that the x force can be represented by +sin, and the y force by +cos, as cos is the adjacent and as the force starts from straight up, then the adjacent is 90°. If it is 105°, then it can be known as x = +cos and y = -sin. This continues around for the rest in resolve.
As all of these forces are on the X and Y directions in the same direction, we can just add them. Then, we use anti resolve which just does what resolve does, only from two forces to one. This then gives it a single force which sums up the two forces, which can then be used.

3.2 From a file

4 One more thing

If you are wondering where your old default text is; it has been stored as a template. The template menu can be used to access and restore it.