# PPP Assignment 2 Initial Specifications

Ivans Saponenko

Bournemouth University SDAGE Y1 2016

PPP Assignment 2 Initial Specifications

## Objects

### PartycleSystem

ParticleSystem class is the main class, which will contain all particles and control their behaviour. Creating particles using MagicCircle shape, deleting them when they go off the screen, ParticleSystem will control the particles amount in the scene. Main job of ParticleSystem will be updating particle velocity by getting velocities from FlowSpace and TheMagicCircle classes. Also ParticleSystem will have a built in physics structure, which will also define particles behaviour. Particle is represented by a point, with mass, velocity and constrain bool.

**Physics structure.** Physics structure will provide constants, which will be used in particle velocity pre-computation. Now it is only air friction and gravity, but I am planning to add more. The reason these constants are in structure is that I want them to be easy accessible and changeable during play-time.

**Final Velocity Algorithm.**

1. GetVel : Getting initial particle velocity
2. PhysicsVec = (GetMass * gravityVec + GetVel) – frictionVec : physics velocity module
3. FlowVelVec = PhysicsVec + FlowVec : flow velocity module
4. FinalVel =

   if( ParticleOnCircle and |FlowVelVec| > |CircleAttractionForce|)

   FinalVec = FlowVelVec + CircleAttractionVec

   Else if ( !ParticleOnCircle and ParticleInCircleRange)

   FinalVec = FlowVelVec + CircleAttractionVec

   Else if (ParticleOnCircle)

   FinalVec = CircleVelocity

   Else FinalVec = FlowVelVec : Circle velocity module

**FlowSpace**

Flow space is a grid with vectors on each of grids vertex. Flow space will imitate movement of air and aerodynamics. Flow space is constructed by dividing screen in equal sqare segments (2d) or or cubes (3d). My implementation for now have issue of vectors overrunning themselves, each of them belongs to each tile, which will cause a lot of bugs, and But I will solve this problem.

**FlowSpace Anatomy.** Flow Space is constructed with array of structures (struct Grid) containing 4 vectors each. Grid tiles have no position, instead they are represented as screen division coordinate, easily located by dividing particle coordinates by number of tiles in space and flooring the division.

**Particle->FlowSpace interaction.** FlowSpace is updated every frame. When particleSystem calls FlowSpace for a FlowVector, FlowSpace will calculate the vector of motion for every particle using one of this algorithms: (choice will be made when I will see how fast those computations will work)

1. Simple
$$\frac{1}{4}\sum_{i=1}^{4} \vec{Fv}_i$$
2. Triangulating the particle distance to the vertex in the tile
$$\sum_{i=1}^{4} \frac{d_i * \vec{Fv}_i}{\frac{1}{4}\sum_{i=1}^{4} d_i}$$
$d_i$ - is a distance from a particle point to the vector i

**Sequence of Computation.** ParticleSystem request -> FlowSpace function finds the tile particle is in -> VectorApproximation Calculation (**in first algorithm case we skip this step, because vector will already exist) -> returning a FlowVector to ParticleSystem for further computations

For V1 I will use the first formula, which will always approximate vector for each tile.

**TheMagicCircle>FlowSpace interaction (starting from V3).** MagicCircle motion will be converted to vector map, which will interact with FlowSpace, and will add vector map vectors to flow space flow vectors, converting motion of the object to motion of the “air”.

**FlowSpace Behaviour.** As a stretch goal the FlowSpace behaviour will be defined by the laws of aerodynamics. I will be introducing new variable called pressure, which will make vectors emit the pressure zones, and point to "low pressure" zones, and away of "high pressure" zones. That will probably make the FlowSpace vectors behave as natural air, creating turbulences and "sucking" effect.

For V1 I am making FlowSpace vectors oscillate in random. This will make particles move as in Brownian movement. I am taking this as a starting point for more complex algorithm implementation.

### The Magic Circle

The Magic Circle is the main interactive object in the game. Circle (or sphere in 3d) is represented by vector position and radius (pos and r).

**Circle Controls.** Circle will take position from the mouse pointer position on the window. Left click will activate "generation" of particles, while in V2 right click will activate vacuum cleaner effect.

**Circle -> Particle behaviour.**

1. Circle will generate particles, which are constrained to circle at first. Player can "shake" them off.

2. Circle will attract particles which will fall in its "attraction range" and create an attraction vector which will be calculated by
   $F(Vector) = powerOfAttraction/Constant^d * VectorOfAttraction$ function.
   This feature was inspired by strange extreme electrostatical dust attraction to my mac.

3. Particles will become constrained to the Circle when they collide with it.

**Circle -> FlowSpace interaction.** TheMagicCircle will interact with FlowSpace by converting its velocity to "air" movement flow vectors.

will be added in V3

**Version Plan**

**V1.**

- Working Particle system

**V2.**

- Improved Working Particle system
- Simple Physics
- Magic Circle “suction” effect
- FlowSpace grid 2nd algorithm

**V3.**

- FlowSpace interaction with Circle
- FlowSpace “pressure” algorithms
- Advanced Physics
- Particle velocity based colouring

**V4.**

- Converting to 3D space

**V5: (stretch).**

- Scene integration
- Particle – Scene interaction

**V6: (überstretch).**

- Particle Shaders

**V7: (lucky one).**

- Working vacuum-machine game

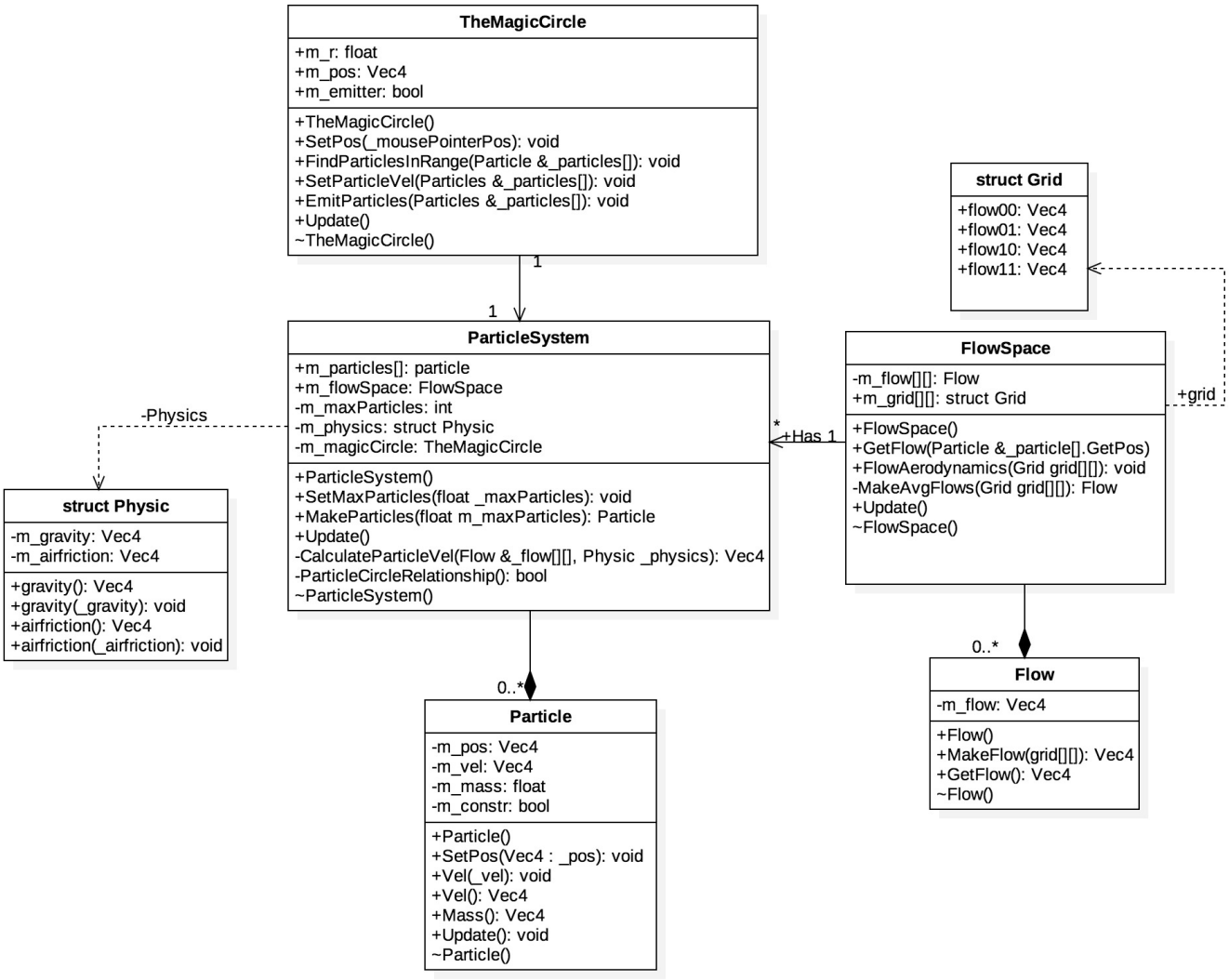


*Figure 1*. UML classes.