Simulation & Animation - SS 2023

Magnity
Game Doc

Aaron Giner, [redacted] aaron.giner@student.tugraz.at
Laura Pessl, [redacted] laura.pessl@student.tugraz.at

June 10, 2023

1 The Game

Our game "Magnity" (Magnet + Gravity) is a single- or 2-player COOP game with the objective of transporting a fragile object through fixed levels to a target destination. The players have to make sure that the object doesn't break by balancing it between two player-controlled magnets that influence the object's behavior. If the objects escape the magnet's grasp or hit a non-"bouncy" wall they break and the players failed the level. The magnets themselves are not influenced by any other objects and only change their position due to the interaction by the user. The players' magnets are restricted in their movement to certain designated areas in the scene that will stay the same within one level. The players can activate or deactivate their magnets at will to influence the pull on the object, which is needed to overcome reach the target area and avoid collisions with walls. To give visual feedback to the player, not active magnets will get greyed out. Additionally, the magnets can have different strengths of gravity giving the user an additional tool to maneuver the object past barriers and across the scene to its destination. The magnets themselves are not solid objects, i.e., the object can pass through the magnets.

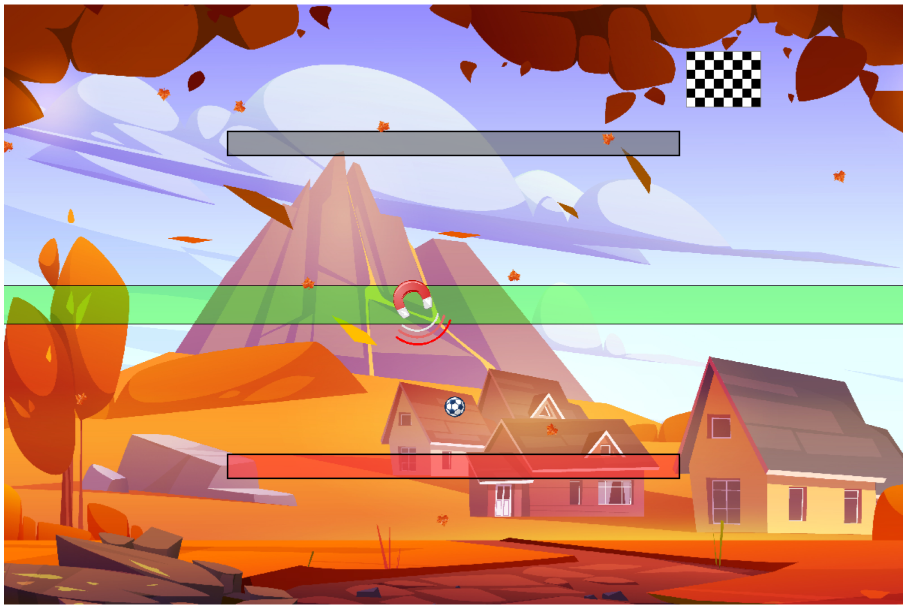

Figure 1: Level layout with the target area, magnet area, walls (red = "bouncy", black = objects shatter)

2 Execution Instructions

The Windows executable is located in the bin/ folder and should be ready to execute. Important: Execute the binary from the project root!

3 User Documentation

3.1 Controls

1. Player 1 movement: W, A, S, D
2. Player 2 movement: Up Arrow, Left Arrow, Down Arrow, Right Arrow
3. Player 1 magnet control: E
4. Player 2 magnet control: ENTER
5. Pause Game: ESCAPE
6. Activate simulation control: ESCAPE + Press Button "Toggle Control Panel"

3.2 Magnets

Magnity is a two-player game that has the main goal of safely maneuvering an object to a designated finish area. To influence the behavior of the object, the users can move around their magnets in the magnets areas with their keyboards arrow keys as well as the W, A, S, and D keys, respectively. The E-key and the ENTER-key can be pressed to deactivate or activate the magnets. The difficulty lies in working together since both players need to move their magnets to the correct positions so that the object does not get out of reach for the magnets. Since an object gravitates towards the stronger or geographically closer magnet, sometimes moving away from the object while the other player moves closer can help direct the object's movements.

3.3 Obstacles

To avoid shattering, players have to take care of black borders. If the object hits such a border, the object will shatter and the game will be over. Red borders on the other hand let the object bounce off, which can come in handy to overcome obstacles.

3.4 Levels and Demos

Following a seasons theme, each level delivers a different gameplay with increasing difficulty. The listed demos can help view the used technical techniques and understand the game play better.

4 Technical Documentation

The following techniques have been implemented into our game.

4.1 Rigid Body Dynamics

Rigid body dynamics are implemented using Verlet Velocity Integration. All rigid bodies store their properties, such as linear and angular momentum, center of mass, torque and applied forces in member variables that get modified in each time step from initial state y0 to the end state yEnd. After the calculations of the properties, a collision detection starts. With the use of boundary boxes, intersections get detected, resolved, and the momentum vectors of all rigid bodies involved, using the impulse formula, get updated. Upon collision spin can be applied to the rigid bodies which influences the torque value as well as the angular momentum. After all computations are finished, the new state yEnd of the rigid body gets stored as it will serve as the next y0 state in the following time step. To visualize the linear momentum vector as well as the angular momentum value, click the button "RB: Show Momentum and Velocity vectors" in the toggle control panel. The code for this technique can be found in "RigidBody.[cpp/h]".

4.2 Voronoi Fracture

Upon collision with certain objects, rigid bodies can result in fractures. The first step is calculating six Voronoi seed points, which have a higher density in immediate proximity to the detected collision point but are completely randomly distributed otherwise. Following this procedure, we loop through all pixels in the objects texture image, find the closest two Voronoi center points and calculate the signed distance from these particular pixels coordinates to the two Voronoi points. Before calculation, a noise filter is applied. If one of them has a negative signed distance, this pixel lays inside this Voronoi cell. Since we create a new image for each Voronoi cell, we take the image this Voronoi point belongs to and colour the pixel with the same colour that the original objects image at this particular point has. After this procedure, new rigid bodies, representing the gained fractures, get created, inheriting the original objects properties. To see the Vornonoi seed points as well as the Voronoi cells push the "VF: Show cells" button in the toggle control panel. Notice, that since our Voronoi fractures get calculated based on the collision point, these are not the real fractures due to their precomputation. Tick or untick the "VF: Use Noise" tickbox to turn the noising on or off. The code for this technique can be found in VoronoiFracture.[cpp/h].

4.3 Particle Dynamics

We used Particle Dynamics as a cosmetic tool to make some of our levels more lively. Specifically, in Level 3, we used Particle Dynamics to move leaves in a wind-like manner. We did this by applying two constant forces, gravity, and a constant wind force, as well as randomly placed particles of high mass, which attract the leaves to a certain extent and give the movement a more realistic look. For Level 4, we did a similar thing, but instead of leaves, we used snowflakes in a winter-themed level.

To implement Particle Dynamics, we created a class "Particle", which has all of the necessary state information of a single particle. We also create a class "ForceSource" which lets define a specific type of force (specifically: Gravity, AntiGravity, and Constant). Finally, we created the class "ParticleDynamics," which handles all of the simulation aspects. In each update step, we update the positions of all of the Particles using either the Runge-Kutta 4 or Euler method. In each step of the RK4 algorithm, we update the positions of all particles and then calculate the force acting on each object for the next integration step.

To toggle a visualization of the trails of all active particles, press the "PD: Draw Trails" button in the control menu. To show a force field of forces acting on locations across the scene, press the "PD: Draw force-field button" in the control menu. You can switch between RK4 and Euler integration using the provided checkbox in the control menu.

All the source code regarding Particle dynamics can be found in "ParticleDynamics.[cpp/h]".

4.4 Path Interpolation

We used Path Interpolation as both a cosmetic feature as well as a game element for Magnity. Firstly, we use splines to move the "target area" across a specified path in some of the levels. We also use splines to move birds as well as butterflies across the scene in levels 1 and 2.

To implement Path Interpolation, we created the classes Spline and SplineSegment. The spline is used to instantiate all of the necessary data structures for interpolation, to update the position, and draw a certain defined Sprite in at the given location. The SplineSegment class consists of the four necessary control points for the Catmull-Rom spline. During initialization or upon moving the control points with the mouse, we initialize/calculate an arc-length table, which we then use to move along the spline.

Press the "PI: Draw Curves" button in the control menu to toggle a visualization for all spline curves in the scene. To toggle the visualization of the control points and arc-length samples, press the "PI: Draw Control points and Arc-length Samples" button in the control menu. You can additionally adjust the easing mode for the movement along the spline as well as the traversal speed.

All the source code regarding Path Interpolation can be found in "PathInterpol.[cpp/h]".