Dość rzeczywista, modyfikowalna symulacja żonglowania

Piotr Kurek

9 czerwca 2020

Spis treści

1 Opis projektu

Właściwie wystarczy sobie wejsc pod adres http://www.gunswap.co/ i już wiadomo mniej więcej na czym polega moja symulacja. Główne różnice to oczywiście mniej opcji w wyborze patternów, ale za to więcej innych parametrów do zmodyfikowania, z zastrzeżeniem, że po zmianie domyślnych ustawień symulacja niekoniecznie będzie działać prawidłowo(projektu raczej nie można określić jako ukończony). Kolejną istotną rożnicą jest to, że moja symulacja nie jest odtwarzana i zapętlana, ale na bieżąco wyliczana i podatna na wiele źmiennych środowiskowych", jako że zrobiłem ją w unity.

2 Dla nieobeznanych z unity

2.1 Piłka

Po pierwsze do symulacji żonglowania potrzebuje piłek, które modeluję jako obiekt posiadajcy $MeshRenderer$(wygląd) w kształcie kuli, $Rigidbody$(umożliwa m. in. nadanie piłce wektora prędkości) oraz dwa $SphereCollider$. Chyba wystarczyłby jeden $SphereCollider$, ale posiadając dwa mogę wyłączyć ten który nie ma zaznaczonej opcji $IsTrigger$ i w ten sposób uzyskuje symulację bez kolizji. Oczywiście trzeba to wykonać na każdej piłce, ale da się to zrobić dwoma kliknięciami, więc nie widziałem konieczności aby to uprościć. Piłka posiada również skrypt $BallCollision$, który jest odpowiedzialny za wykrycie i poinformowanie o kolizji z innymi piłkami.

2.2 Dłoń

W projekcie nazwałem te obiekty odpowiednio $Left/RightHand$, posiada niezbyt duży $SphereCollider$, który umożliwa natychmiastowe złapanie piłki, gdy ta znajdzie się wewnątrz, oczywiście potrzebowłem to odpowiednio zakodować w skrypcie $HandlingBall$, który ten obiekt również posiada. Drugą główną funkcją, którą pełni ten skrypt jest oczywyiście odpowiednio ztimingowany wyrzut piłki.

2.3 Ręka

Obiekt nazwany Arm, zawiera tyllko $MeshRenderer$, jego główną funkcją jest to, że obiekt, którego opiszę poniżej go obraca, a wraz z nim dłoń, ponieważ dłoń jest do ręki "przyczepiona"jako $ChildObject$

2.4 Punkt obracający

W projekcie nazwałem te obiekty odpowiednio $Left/RightSide$. Posiada skrypt $RotationPattern$, którego główną funkcją jest obracanie tegoż punktu. Nie jest to jednak, aż tak proste ponieważ jednoczesny obrót wokół osi y oraz z powoduje również zmianę rotacji x, co sprawia, że rotacja nie zapętla się tak jak powinna.

W związku z tym, każdy punkt obracający obraca dwa dummy sześciany, jednego wokół osi y, drugiego wokół osi z, a następnie ustawia swoją rotację jako połączenie rotacji tych dwóch sześcianów. Oprócz tego skrypt umożliwa rozróżnie tego, która strona jest lewa, a która prawa w celu ztimingowania całej rotacji.

2.5 Kamera

Domyślnie jest jedna w każdym projekcie unity. W tym projekcie dodałem to niej skrypt $PatternManager$. Odpowiada on za możliwość specyfikacji parametrów danego patternu, na ich podstawie wylicza nieco innych, dzięki czemu może ustawić pozostałym skryptom odpowiednie wartości. Generalnie sporo wychodzi tych parametrów, ale prawie wszystko co się z nimi dzieje jest bardzo proste, niektóre zmienne są redundantne, ale zwiększają czytelność i możliwe, że w bardziej skomplikowanych patternach się przydadzą.

3 Jak to mniej więcej działa

3.1 Pattern Manager

Na początku należy obliczyć i ustawić wszelkie niezbędne parametry, jeszcze przed wykonaniem się właściwej części symulacji. Dlatego używam funkcji *Awake*, które jest zawsze wykonywana jako pierwsza. Kolejno:

1. Ustalam czy mam do czynienia z patternem parzystym czy nieparzystym i podstawiam odpowiedni wektor.
2. SetCorrectVector() - Powiększam część składową y w zależności od parametru *patternSpeed*, skalując odpowiednio część z, tak aby piłka pokonała tą samą odległość z.
3. SetCorrectSpeed() - W zależności od *patternSpeed* i parametru $ballMaths$, który zależy od ilości piłek i rodzaju patternu wyliczam odpowiednie bazowe parametry rotacji.
4. SetPatternRotation() - W zależności od ręki i patternu rotacja, a czasem również wektor musi się troche różnić, więc ustawiam odpowiednie wartości biorąc te rzeczy pod uwagę
5. SetUpPatternStats() - Ustawiam parametry, które potrzebują lewo/prawo ręczne skrypty $HandlingBall$.
6. DistributeBalls() - Rozdaje każdej dłoni odpowiednią ilość piłek.
7. Na koniec jeszcze ewentualnie wyliczam odpowiedni *timeScale*, oraz w $Update()$ na bieżąco śledzę ilość wszystkich zderzeń danego podejścia, aby mieć wszystkie przydatne informacje w jednym miejscu w inspektorze.

3.2 RotationPattern

Następnie należy wprawić ręce w ruch, przechodzimy, więc do skryptu odpowiedzialnego za tę właśnie czynność.

1. Dopasowywuje timing rotacji do ustawień początkowych
2. $StartCoroutine(hand.BallRelease(hand.initialDelay))$ - Lewą rękę będę obracać od początku, więc od razu informuje lewą dłoń kiedy ma zacząć wyrzucać piłki.
3. $StartCoroutine(SetUpTiming())$ - Prawa ręka odczekuje połowę okresu rotacji i następnie robi to co lewa.
4. $if(throwMode)$ - Jeśli mam do czynienia z lewą ręką lub upłyneła odpowiednia ilość czasu, zaczynam ciągłą rotację, ponieważ całość jest w $FixedUpdate$. Obracam kolejno przykładowy obiekt wokół osi y ($yVec.Rotate(0f, rotYSpeed * Time.fixedDeltaTime * Mathf.Cos(yTime * PatternManager.COS_SCALE), 0f);$),

drugi analogicznie wokół osi z, a następnie ustawiam na obiekcie do którego podpięty jest ten skrypt sumę ich rotacji - $transform.rotation = Quaternion.Euler(0f, yVec.rotation.y * 100, zVec.rotation.z * 100)$.

5. Funkcja cosinus w zależności od tego na jakim etapie jest dana składowa rotacji zapewnia jej kolisty kształt, zahardkodowane mnożenie razy 100 jest tam dlatego, że suma tych rotacji, mimo że raczej zachowuje odpowiednie proporcje, to jest dziwnie pomniejszona i należy ją odpowiednio zwiększyć.

6. $TimeRotation()$ - Metoda odpowiedzialna za właściwy kształt i zapętlanie się rotacji.

3.3 HandlingBall

Skrypt dbający o poprawne zachowanie piłek. Zaczyna działać w tym samym momencie co *RotationPattern*.

1. Wyciąga odpowiednie referencje z pierwszej piłki przydzielonej przez $PatternManager$ i cały czas ustawia jej pozycja taką samą jak obiekt do którego podpięty jest skrypt(dłoń).

2. Wcześniejszy skrypt odpowiada za początkowy timing, uruchamia on $IEnumeratorBallRelease()$, typ funkcji $IEnumerator$ pozwala w niej odczekać pewną ilość czasu i dopiero później wykonać instrukcje.

3. $scale = ball.transform.position.y - basicPos.y$ - Wylicza ewentualne wydłużenie, lub skrócenie *scale* pojedyńczego czasu oczekiwania na piłkę

4. $Vector3 fixedVetor = newVector3(properVector.x, properVector.y * (1 + Random.Range(-maxErrorY, maxErrorY)), properVector.z * (1 + Random.Range(-maxErrorZ, max$ - losowo zmienia domyślny wektor, tak aby zawierał on pewien losowy, ograniczony procentowo błąd.

5. $ballRb.velocity = fixedVetor$ - ustawia prędkość piłki na równą wyliczonemu wektorowi.

6. $if(balls.Count - 1 > ballNum)$ - jeśli wciąż są jakieś nieużywane piłki, które zostały przydzielone danej dłoni, to następnie przygotowywujemy kolejną z nich.

7. $StartCoroutine(BallRelease(patternTime/(1 + scale)))$ Zapętla wyrzucanie piłki.

8. $OnTriggerEnter$ - odpowiedzialne za złapanie pobliskiej piłki i wyciągnięcie z niej odpowiednich referencji.

4 Parametry Pattern Managera, które raczej można edytować

1. adjustSimulationSpeed - Czy zmienna poniżej powinna mieć zastosowanie.
2. simulationSpeed - Jak szybko ma się wykonywać prędkość rotacji niezależnie od wszystkiego(odpowiednio zmieniam wartość timeScale aby to uzyskać, jest to wydaje mi się konieczne do w miarę obiektywnych testów.
3. patternType - wpisanie Reverse Cascade zmienia styl żonglowania na właśnie taki, w przeciwnym razie domyślny
4. ballsInPattern - Ilość piłek w patternie.
5. patternSpeed - Szybkość rotacji rąk/wysokość rzutów, jest to jedyny czynnik który wpływa na wysokość, ale nie jedyny który w wpływa na szybkość rotacji.
6. maxErrorZ - Maksymalny procentowy błąd składowej z wektora wyrzutu.
7. maxErrorY - Maksymalny procentowy błąd składowej y wektora wyrzutu.
8. timeScale - Procentowe skalowanie czasu, 1 to prędkość normalna, 2 dwa razy szybsza, 0.5 dwa razy wolniejsza.

5 Parametry Pattern Managera, które na obecną chwilę nie zaleca się edytować

1. correctCycle - Potrzebne do odpowiedniego timingu patternu w początkowej fazie.
2. cosScale - Odpowiednia wartość pozwala uzyskać kolisty ruch rąk.
3. offset - Obecnie nie używane w projekcie, wprowadzone swego czasu ze względu na niedokładność przy odpowiednio dużej prędkości rotacji.
4. basicVector - Wektor wyrzutu piłki, odpowiednio ustawiany na jedną z dwóch wartości poniżej w zależności od liczby piłek.
5. basicEvenPatternVector - Wektor rzutu do tej samej ręki.
6. basicOddPatternVector - Wektor rzutu do ręki przeciwnej.
7. basicHandTime - Połowa okresu jednej rotacji. Używam, gdy odnoszę się do czasu, który piłka spędza w dłoni.
8. basicAirTime - Połowa okresu jednej rotacji. Używam, gdy odnoszę się do czasu, który piłka spędza w powietrzu.
9. basicRotYAmount - Wielkość kąta rotacji Y

10. basicRotZAmount - Wielkość kąta rotacji Z

11. basicROT_Y_TIME - Połowa okresu jednej rotacji.Używam, gdy odnoszę się do czasu rotacji wokół osi yY.

12. basicROT_Z_TIME - Połowa okresu jednej rotacji.Używam, gdy odnoszę się do czasu rotacji wokół osi Z.

6 Eksperymenty

Eksperymenty miały na celu znaleźć mniej więcej minimalna wymaganą precyzje, aby dało się przez pewien czas żonglować dany pattern oraz optymalną dla niego wysokość. Generalnie próbowałem wysokości, które wydawały mi się pasować, starałem się jak najbardziej zwiększyć błąd z którym moja symulacja wciąż sobie radzi. Radzi tj. ma sporą szansę na wykonanie ok. 20-50 złapań.

6.1 Wyniki

Wysokość liczę ze wzoru = (7.1/patternSpeed)/9.81 * (7.1/patternSpeed)/2
7.1 to wartość y bazowego wektora
9.81 to przyspieszenie grawitacyjne

1. 5 piłek, błąd maksymalny 0.05, pattern speed 1.25-1.4, wysokość to przedział 1.31-1.64m
2. 7 piłek, błąd maksymalny 0.036, pattern speed 1.2-1.3, wysokość to przedział 1.52-1.78m
3. 9 piłek, błąd maksymalny 0.027, pattern speed 1-1.05, wyskość to przedział 2.33-2.57m

Uważam, że jest dość prawdopodobne, że wyniki mają zastosowanie w rzeczywistości, ale nie należy im do końca ufać, ponieważ nie poświęciłem na testy dostatecznie dużo czasu i zastosowana metoda jest z pewnością podatna na przekłamania.