Sarah Steenhuis, 3018188
Kernmodule 1, Documentatie

Toelichting UML
Ik had begrepen dat op de class diagram alleen de informatie hoeft te staan die duidelijk maakt hoe het systeem werkt. Daarom heb ik niet alle functies en variabelen er in verwerkt.

De belangrijkste classes zijn de Snake, SnakeHead en Mouse. De Mouse is het eten en zorgt dat de speler een doel heeft. De Snake zorgt dat voor elk lichaamsdeel ook echt een object gemaakt wordt. De SnakeHead beheert de richting, lengte en positie van de slang en bepaalt tevens waar de bullet geactiveerd wordt. De bullet is ook belangrijk voor de gameplay. Hij doet zelf niet veel meer dan de richting van de SnakeHead overnemen.

Ik moet eerlijk toegeven dat ik die lijntjes tussen de classes nog best lastig vind. De huidige lijntjes heb ik vooral gekozen door de dingen af te strepen die ik helemaal niet toepasselijk vond. Veel van de lijntjes zijn slechts associaties: verbanden die nodig zijn voor het spelen, maar niet nodig om te bestaan. De bullet kan namelijk een Snake of een Mouse dood maken, maar is verder niet nodig voor één van deze twee om te bestaan. De Snake weet niet wat hij moet doen zonder de SnakeHead, maar de SnakeHead bestaat niet uit stukjes Snake. Er is ook geen sprake van Inheritance.

De scene flow is redelijk gelijk gebleven aan hoe ik het in mijn activity diagram had gepland. Het enige verschil is dat je vanaf ‘Game Over’ nu ook direct weer een game kan starten en dat er geen options zijn, maar een rules canvas met alleen uitleg.

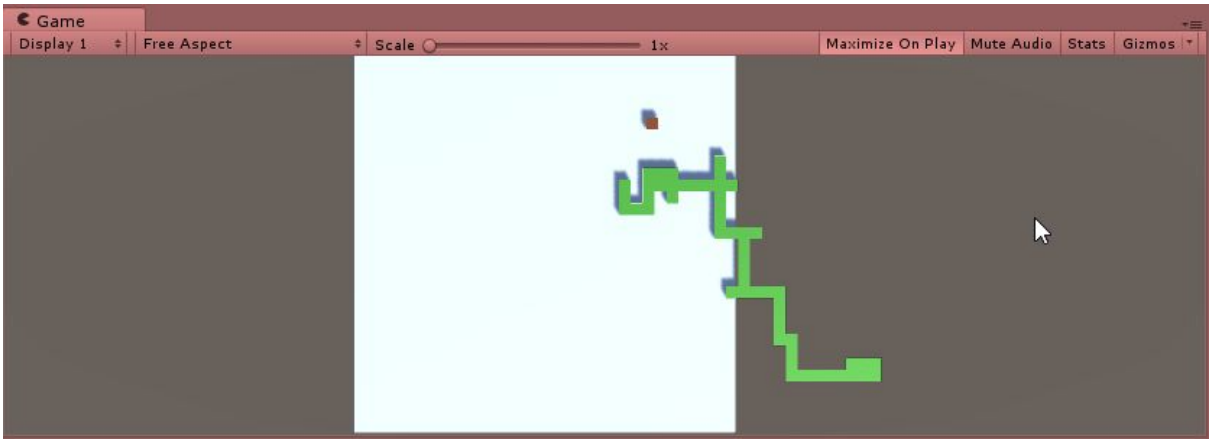
Waarom heb je voor dit ontwerp gekozen?
Het eerste wat ik werkend wilde hebben was een delegate met een event waarin alles zou gebeuren wat moest gebeuren als de muis geraakt werd door de speler (en later de bullet van de speler). Dit was verbazingwekkend makkelijk om te implementeren en zorgt voor makkelijke communicatie tussen de muis, snake, en later ook bullet. De muis weet wanneer hij een nieuwe positie aan moet nemen, de snake weet dat hij langer moet worden, en de bullet weet dat hij gedeactiveerd moet worden allemaal dankzij één event.

Een deel van het ontwerp komt van een youtube tutorial. De GetNext en SetNext van de slang bijvoorbeeld. Deze functies zijn nodig voor de manier waarop de slang in deze opbouw voortbeweegt. Ik heb deze oplossing laten staan omdat mijn eigen pogingen niet werkten.

Ik had al snel het besef dat er veel van dezelfde objecten gebruikt zouden worden voor het slangenlijf. Een ObjectPool was daarom vanzelfsprekend. Het script zorgde er zelfs eerst voor dat er op elke verandering een gameobject instantiated werd en eentje verwijderd. Dat is natuurlijk niet erg efficient. Daarom heb ik daar een ObjectPool geïmplementeerd.

De ObjectPool heb ik vervolgens hergebruikt voor de bullets. Deze zijn lang niet altijd allemaal op het scherm en kunnen daardoor makkelijk hergebruikt worden voor meer efficiëntie.

Hoe is het ontwerp door het project heen verandert en waar kwam dat door?
In eerste instantie wilde ik het hele script helemaal zelf schrijven. Ik was redelijk op weg: ik had een spel waarbij je een blokje was welke bij contact met een ander blokje een event activeerde. In dit event wilde ik dan zorgen dat de speler langer kon worden.
Vervolgens had ik echter problemen met het toevoegen van een staart. Het maken van een child die altijd één stukje verder van de kern af stond was op de een of andere manier heel moeilijk. Ik gaf aan dat hij alleen de z positie moest aanpassen, en toch werd soms de x positie aangepast. Ik kwam hier niet mee verder en ben daarom een tutorial gaan zoeken. Van deze tutorial heb ik de beweging van de snake overgenomen en geoptimaliseerd. Ik heb het doodgaan van de muis laten staan, omdat dit prima bleef functioneren.



De maker van de tutorial maakte geen optimale code. Allereerst werkte hij vanuit een GameManager class. Dit is op zich niet fout, maar kan onduidelijkheid scheppen omdat je dingen met de Snake doet in een class die niks met de snake te maken heeft. Ik heb hiervoor in de plaats een SnakeHead script gebruikt. Ik heb ook geprobeerd om alles vanuit de snake te doen, maar dit zorgde er voor dat de AddBody functie voor elk actieve lichaamsdeel aan de delegate werd toegevoegd. Hierdoor verdubbelde de lengte steeds bij het scoren van een punt, wat niet de bedoeling was. Alles regelen vanuit één Head object bleek de meest efficiënte manier.

```
26      //Direction management
27      public enum Direction {
28          Up,
29          Down,
30          Left,
31          Right,
32      }
33      private Direction direction;
34      private Vector3 nextPos;
35
36      //these vectors are used to move in the 4 different directions
37      private Vector3 goUp = new Vector3(0, 0, 1);
38      private Vector3 goDown = new Vector3(0, 0, -1);
39      private Vector3 goLeft = new Vector3(-1, 0, 0);
40      private Vector3 goRight = new Vector3(1, 0, 0);
```

Ook gebruikte deze persoon een integer om de richting bij te houden. Ik heb in plaats hiervan een enum gebruikt, omdat je deze een naam kan geven. Dat maakt de code beter leesbaar.

Als laatste maakte deze man in zijn code op elk frame een hoop nieuwe vectoren. Ik heb deze in variabelen gestopt zodat ze maar één keer gemaakt hoefden te worden en daarna vrij gebruikt kunnen worden.

Nog een iets minder belangrijke aanpassing: het leek mij een slim idee om van mijn SceneManager een singleton te maken, omdat hier altijd maar één van hoefde te zijn. Hierdoor ging er echter een link in een button verloren in het GameOver scherm, waardoor je de knoppen daar niet meer kon gebruiken. Het al bestaande SceneManager object verwijderde het object waar ik de button aan had gelinkt. In dit geval was de Singleton dus inefficiënt, en heb ik hem weer weggehaald.

Wat ga je volgende keer anders aanpakken?
Virtual Object Pool. Ik heb twee object pools die vrijwel hetzelfde doen. Die zouden best wat code uit één overlappende object pool class kunnen halen.

Ook wil ik me nog meer verdiepen in het verminderen van het gebruik van MonoBehaviour functies en dependencies. Op het moment maak ik op meerdere plekken gebruik van de Update functie. Eigenlijk zou ik dat willen verminderen - mits dit mogelijk is.

Waar ben je niet aan toegekomen?
Screen wrapping. Hier heb ik wel een poging tot gedaan, maar dit zorgde voor nieuwe problemen omdat ik hiermee ook de actieve gameobjects ook steeds wilde laten verplaatsen in plaats van op elk frame een nieuwe laden. Ook kan het voedsel nu in principe buiten het scherm spawnen als de coördinaten dat toestaan. Hier kwam ik pas achter in de build, want hierbij is de resolutie heel anders dan in de editor. Toen ik dit probeerde te fixen door met een renderer te checken of de muis wel binnen het scherm was gespawnd, crashte het spel helemaal. Dat heb ik daarom maar weer weggehaald.

In het begin leek het me ook tof om iets te doen met verschillende moeilijkheidsgraden. Op easy gaat de slang dan bijvoorbeeld langzamer dan op hard, of het voedsel loopt van je weg op hard. Hier ben ik verder niet mee bezig geweest, omdat ik steeds tegen andere problemen liep.

Ik heb ook nog het idee gehad om naast het voedsel ook de kogels zo te maken dat je ze kan oppakken, of dat je je lichaamsdelen afschiet. Ik wilde echter eerst de basis hebben staan, en dat was een Snake waarbij je het voedsel moet schieten om punten te scoren. In mijn vrije tijd kan ik nog met deze extra twists verder.