**Lerend algoritme**

**“Hoe een haas een koe vangt”**

**Opdracht5: Genetisch algoritme**

|  |  |
| --- | --- |
| 1. Maak een standalone (.exe) applicatie, die op een windowsmachine probleemloos start. | Een .exe applicatie wordt gemaakt (KMINTWeek5\SDLFramework\Debug) wanneer de Visual Studio applicatie gedebugged wordt. |
| 2. Gebruik de kennis over statistical FSM (opd 3) en Autonomously Moving Game Agents (opd 4) | ProbabilityDistribution.cpp, StateMachine.h |
| 3. Op je scherm zie je (tegelijk!) 4 “*instanties*”: | Arena.cpp |
| 4. Eén *instantie* bestaat uit 1 Koe, 1 Haas, 1 Pil en 1 Wapen object. | InstanceOfGameEntities.cpp |
| 5. Elke *instantie* heeft een eigen kleur (Rood, Groen, Blauw en Geel). Wat betekent dat alle plaatjes (objecten) uit die *instantie* een ‘waas’ in die kleur over zich heen hebben. | Arena.cpp |
| 6. Objecten binnen één *instantie* hebben wél interactie met elkaar (later meer hier over), maar nooit met de objecten uit een andere *instantie* (visueel: die met een andere kleur). | InstanceOfGameEntities.cpp |
| 7. Alle *instanties* zijn totaal gelijk qua regels, op twee zaken na: |  |
| a. De kleur (“waas”) van de sprites. | InstanceOfGameEntities.cpp |
| b. De kansverdeling van de state-transities@ voor de koe. | ProbabilityDistribution.cpp |
| 8. Het volgende is van toepassing op de “wereld” (het hele veld / scherm): |  |
| a. De wereld is 800 pixels breed en is een toroid. | FWApplication.cpp |
| b. De wereld bevat 2 ‘spawnpunten’ die horizontaal altijd 400 pixels uit elkaar liggen (let op toroid!). 1 voor de koe en 1 voor de haas. | InstanceOfGameEntities.cpp |
| c. De haas is twee keer zo snel als de koe. | InstanceOfGameEntities.cpp |
| d. De haas is **altijd** alleen maar aan het chasen. | InstanceOfGameEntities.cpp |
| e. Bij het uitvoeren van de applicatie worden 5 **runs** uitgevoerd. Elke run duurt 30 seconden. Na 2.5 minuut is het programma dus klaar. | Arena.cpp |

|  |  |
| --- | --- |
| 9. Het volgende is van toepassing op elke **run**: |  |
| a. In die 30 seconden zie je de 4 instanties animeren, dus 4 koeien, etc. (zie eis ‘3’ t/m ‘7). | Arena.cpp |
| b. De koe start met wanderen | InstanceOfGameEntities.cpp |
| c. Als de haas binnen 300 pixels komt, gaat de koe 1 van de volgende dingen doen: | CowWanderingState.cpp |
| i. Vluchten (flee volgens boek. Als de haas de koe heeft gevangen, krijgt de haas 10 punten en worden de koe en de haas gerespawned.) | RabbitPursuitState.cpp |
| ii. Vluchten & pil zoeken (Zodra pil gepakt: Pil spawned ergens in het veld. Koe blijft vanaf nu staan en is deze hele run voor altijd onkwetsbaar. Haas krijgt geen punten meer. Koe krijgt 1 punt als haas haar raakt. Beesten worden nog wél gerespawned na raken.) | CowSearchPillState.cpp |
| iii. Verstoppen (Koe blijft stilstaan. Koe en haas krijgen geen punten als haas haar raakt. Beesten worden wél gerespawned na raken. Koe is dan niet meer verstopt.)) | CowHidingState.cpp |

**-**

**7**

|  |  |
| --- | --- |
|  |  |
|  |  |
| iv. Vluchten & Wapen zoeken (Zodra wapen gepakt: Wapen spawned ergens in het veld. Zie verder verstoppen.) | CowSearchForWeaponState.cpp |
| d. Aan het einde van elke run (30 sec)… |  |
| i. Worden alle states weer gereset. (Koe is niet meer onkwetsbaar etc.) | InstanceOfGameEntities.cpp |
| ii. Wordt berekend welke *instantie* het meest effectief was voor de koe. **LOG DIT!** | Arena.cpp (Logging in Console Window) |
| iii. Gaat jouw genetische algoritme 4 nieuwe *instanties* maken (een nieuwe generatie) met nieuwe kansverdeling van de state-transities voor de koe. **LOG DIT!** | GeneticAlgorithm.cpp (Logging in Console Window) |
|  |  |