Timing

Taakgroep: Events

Context

Je bent aangekomen bij level 5 en aan de buitenkant is de simulator nog erg saai. Als je de simulator aan familie of vrienden zou laten zien is de reactie hoogstwaarschijnlijk "Is dit waar je zo druk mee bent geweest?!". Dit is vaak het probleem met software ontwikkeling, mensen willen snel resultaat zien. Een goede voorbereiding en zorgvuldige realisatie levert op de lange termijn juist tijdwinst op.

In deze level ga je eindelijk aan de simulatie werken. Met behulp van een timer en events ga je stap voor stap de deelnemers over de baan bewegen. Daarnaast ga je hierover verschillende gegevens opslaan, denk aan bv rondetijden.

Taak: Timing

Om te beginnen ga je een `timer` object aanmaken. Dit object kan je prima aanmaken in een willekeurige klasse. Vanuit de SOLID principes is dit niet wenselijk. De klasse `Race` heeft als doel om de race te simuleren. Daarom is het verstandig om daar ook de `timer` object aan te maken.

Belangrijk om te weten dat je gebruik gaat maken van de `Timer` uit de `System.Timers` namespace.

Aanpak

- Maak in de klasse `Race` een private variable van het type `Timer` en initialiseer deze in de constructor. Geef als interval de waarde `500` (dit staat voor 0.5 seconden).

  Zoals je wellicht hebt gelezen bij de ondersteunende informatie `raised` de timer een `event` met de naam `Elapsed` wanneer de interval verlopen is. Je gaat van deze `event` gebruik maken om de deelnemers in beweging te krijgen.

  Om gebruik te maken van een `event` maak je een `event handler` aan. Dit zijn methodes met als parameters een `object` en een `EventArgs`. Deze methodes beginnen meestal met het woord `On`.

- Maak een `event handler` voor de aangemaakte timer. Geef deze methode de naam `OnTimedEvent`.

  Nu je de timer en een event handler hebt aangemaakt moet je ze alleen nog aan elkaar binden.

- Breid de constructor verder uit en voeg de aangemaakte event handler `OnTimedEvent` toe aan `Elapsed` van de timer.

De timer gaat niet uit zichzelf beginnen en dat is maar goed ook. Je wilt hier zelf controle over hebben.

- Schrijf de methode `Start`. Deze methode heeft geen parameters of returnvalue. Start de timer in deze methode.

Ondersteunende informatie

Alle benodigde informatie over de `timer` is hier te lezen.
Voor uitleg en verdieping over `events` en `event handlers` kan je hier vinden.

Taak: Timed Visualisation

Bij de vorige taak heb je een timer aangemaakt om de deelnemers te bewegen. Om efficiënt om te gaan met de beschikbare resources van je computer ga je het scherm alleen updaten wanneer de deelnemer daadwerkelijk bewogen is. Hiervoor ga je een eigen event voor aanmaken.

Daarnaast gebruik je nog een test `Track` voor een visualisatie. Dit ga je in deze taak aanpassen zodat je de `Track` visualiseert van huidige race.

Aanpak

- In de klasse `Race` maak de event `DriversChanged` aan.

  Nu de event aangemaakt is kan je er gebruik van gaan maken. Het verschil met de event uit de vorige level is dat je nu iets nodig hebt van het event, namelijk de `Track` die veranderd is. Normaal heeft een event handler methode naast `object` een `EventArgs` als parameter. Door gebruik te maken van `inheritance` kan je zelf een klasse schrijven die je vervolgens kan gebruiken als `EventArgs`.

- In je model project maak een klasse `DriversChangedEventArgs` aan en overerf van `EventArgs`. Geef deze klasse een extra property van het type `Track`.

  Nu kan je de event handler methode schrijven voor `DriversChanged`

- Bij de ontwikkelde visualisatie klasse uit level 4, voeg een `event handler` methode toe voor de net aangemaakte event `DriversChanged` waarbij je `DriversChangedEventArgs` als parameter gebruikt.
- Roep, vanuit de aangemaakte event handler methode, de `DrawTrack` aan. Gebruik de `Track` property van de `DriversChangedEventArgs` parameter als input `DrawTrack`.

  Laatste stap is om de event handler te koppelen aan de huidige race. In level 2 heb je hiervoor een property `CurrentRace` aangemaakt.

- Voeg nu de aangemaakte `event handler` toe aan de event `DriversChanged` van de `CurrentRace` property. Zorg ervoor dat de gekozen oplossing bij de static `main` wel aangeroepen wordt.

  Je merkt nu wel dat events moeilijk te volgen zijn in je code. Wanneer je de klasse `Race` bekijkt zie je nergens staan welke `event handlers` gekoppeld zijn hieraan. Dit is een typisch voorbeeld voor een sequence diagram in een technisch ontwerp. Zo'n diagram kan meer inzicht geven in je code en de werking hiervan.

Ondersteunende informatie

Informatie over `inheritance` is hier te lezen
Details over `EventArgs` kan je hier vinden

Taak: Moving around

Zoek een rustig plekje, regel wat versnaperingen en zet een lekker muziekje op. Eindelijk is het zover, je mag de deelnemers in beweging zetten. Deze taak vergt inzicht, creativiteit en doorzettingsvermogen van je. Gebruik de tooling die je tot je beschikking hebt en run je applicatie lekker vaak om meer inzicht te krijgen over de werking van je code.

Aanpak

Voordat je aan de slag gaat is het belangrijk om een aantal dingen vast te stellen. Als je het klassendiagram bekijkt zie je dat `IEquipment` 4 properties heeft.

Bedenk een getal dat je gaat gebruiken als lengte voor elke sectie. Nadat je dat hebt gedaan bedenk dan een formule voor de snelheid van een deelnemer, gebruik hiervoor de properties van `IEquipment`. De deelnemer zal bij elke timer event zich met die snelheid verplaatsen.

Een voorbeeld:
Timer event van 500 (dus een 0.5 seconden)
Lengte van een sectie van 100 (meter)
Snelheid van een deelnemer is Performance * Speed

Wanneer de performance 2 is en de speed 10 zal de deelnemer elke 0.5 zich 20 (meter) verplaatsen. Als de deelnemer 100 (meter) op een sectie afgelegd heeft komt de deelnemer op de volgende sectie.

Zoals je ziet zijn er een aantal waarden waarmee je kan spelen. Wanneer je code goed is kan het toch verkeerd eruit zien doordat je waarden niet goed op elkaar aansluiten, bijvoorbeeld door een te hoge snelheid voor de deelnemers.

- Breid de klasse `Race` verder uit waarbij je de deelnemers bij elke timer event verplaatsen. Tip: vergeet niet de `DriversChanged` event te gebruiken in je oplossing.

  Wanneer je deelnemers netjes bewegen over je scherm mag je oprecht trots op jezelf zijn! Om te voorkomen dat het per ongeluk stuk gaat:

- Schrijf nieuwe tests om je afgesproken code coverage weer te halen.

  Nu er tests aanwezig zijn gaat je werk niet onbewust stuk. Sla je werk op en ga even rustig iets voor jezelf doen voordat je met de volgende level aan de slag gaat.

Taak: Finishing a race

De deelnemers racen lekker rond op de baan. Zolang jij niet op het kruisje drukt gaat dit eindeloos door. Toch kent elke race een einde en dus zal jouw race ook een einde moeten krijgen. Tijdens deze taak ga je een race een einde geven.

Aanpak

Om te beginnen zal je moeten nadenken hoeveel rondjes jij genoeg vindt. De eis is wel dat je minimaal 2 rondjes moet simuleren per race, maar het mag uiteraard ook meer zijn. Om vast te kunnen stellen hoeveel rondjes een deelnemer al heeft gehad zul je dit moeten bijhouden. Normaal gesproken heeft een deelnemer een hele ronde gehad wanneer de deelnemer over de finish gaat.

- Breid de klasse `Race` verder uit waarbij je per deelnemer bijhoudt hoeveel rondjes die al gehad heeft.

  Deelnemers die alle rondjes hebben gedaan moeten niet meer op de baan zijn. Dan zijn ze alle deelnemers tot last.

- Wanneer de deelnemer alle rondjes heeft gehad haal dan de deelnemer van de baan.

  Als je nu de race laat lopen zullen uiteindelijk alle deelnemers van de baan afgehaald worden en kijk je naar een lege baan. Dit is het ideale moment om de race te beëindigen en een nieuwe race te starten.

Taak: Cleaning up

Voordat je aan een nieuwe race begint moet je eerst de oude race correct afsluiten. In deze taak ga je hier invulling aan geven.

Aanpak

Wanneer je wel eens geprogrammeerd hebt in talen zoals C en C++ zal je het zijn opgevallen dat C# veel, vaak op de achtergrond, voor je regelt. Wellicht wel het belangrijkste is wel de `garbage collector`. De `garbage collector` ruimte de dingen op die je niet meer gebruikt waardoor er weer geheugen vrijkomt voor andere dingen. Hoe weet de `garbage collector` of je een object niet meer gebruikt? In het kort, wanneer er geen referenties meer zijn naar het object.

Jouw code heeft nu een referentie tussen de visualisatie klasse en de klasse `Race`. Dit is ontstaan toen je de event handler van je visualisatie klasse hebt gekoppeld aan de `DriversChanged` event van de klasse `Race`. Afhankelijk van jouw code heb je een `strong reference` of een `weak reference` gemaakt tussen de twee klassen.

Wanneer je elke keer een nieuwe `Race` initialiseert en de event handler weer koppelt aan de bijbehorende `DriversChanged` event blijft er een referentie naar het vorige `Race` object. Hierdoor zal de `garbage collector` de oude `Race` object niet opschonen en heb je te maken met een `memory leak`. Je programma gebruikt steeds meer geheugen bij elke nieuwe race. Dit gaat door totdat je geheugen op is en dan krijg je een `out of memory` foutmelding.

Nu je hiervan bewust bent kan je hier in de toekomst rekening mee houden nog voordat je begint met de realisatie. Een mogelijke oplossing voor dit specifieke probleem is gebruik te maken van het `Weak Event Pattern`.

Voor nu ga je het probleem eenvoudiger oplossen. Uiteindelijk gaat het maar om 1 event dus zou het implementeren van het `weak event pattern` enigszins overkill zijn.

- In de klasse `Race` implementeer een methode waarin je alle event handlers van het `DriversChanged` event opruimt.

Nu je alle oude referenties hebt opgeschoond kan je veilig een nieuwe race beginnen.

Ondersteunende informatie

Wil je verdiepen in de `Garbage collector` van C# lees dan dit
Benieuwd wat `memory leaks` in het algemeen zijn? Open dan deze link
Informatie over `weak references` en `strong references` kan je hier vinden.
Meer informatie over het `weak event pattern` is hier te lezen

Taak: Next race

De deelnemers zijn klaar met de race. Jij hebt alles opgeruimd. Het is tijd voor een nieuwe race. In deze taak ga je precies dat doen, een nieuwe race beginnen.

Aanpak

Als je alle voorgaande levels goed hebt doorlopen heb je in de klasse `Data` een property `CurrentRace` en de methode `NextRace`. Wellicht maak je hier al gebruik van. Wat er dus moet gebeuren is dat de methode `NextRace` aangeroepen moet worden wanneer de race voorbij is, simpel toch?

- Implementeer in de klasse `Data` en klasse `Race` een oplossing om een nieuwe race te starten wanneer de vorige race afgelopen is. Tip: gebruik de methode `NextRace` en events. Vergeet niet de oude race op te ruimen.

  Wanneer je nu de applicatie laat draaien zal je merken dat de nieuwe baan niet getoond wordt wanneer de race voorbij is. Dit komt doordat je wel de oude referenties hebt opgeruimd maar nog niet je event handlers van je visualisatie klasse hebt gekoppeld aan de nieuwe `DriversChanged` event.

- Implementeer een oplossing in de klasse `Race` en de visualisatie klasse waarbij de event handler van visualisatie klasse gekoppeld wordt nadat er een nieuwe `Race` is aangemaakt bij `CurrentRace`. Tip: Gebruik een nieuwe event, event handler met eventueel een nieuwe klasse welke overerft van `EventArgs`.

  Eindelijk is alles gedaan. Als je alles goed hebt gedaan wordt er op alle banen achtereenvolgend geracet en is dit zichtbaar op je console applicatie.

Taak: Failing equipment

Waarschijnlijk hebben de deelnemers al de nodige rondjes geraced. Normaal gesproken wil het nog wel eens gebeuren dat er iets stuk gaat. De deelnemer kan dan niet verder en moet dan toezien hoe andere deelnemers de verworven plek innemen. Met wat geluk is dit zo gerepareerd en kan de deelnemer weer verder.

Het doel van de race simulator is om verschillende aspecten van een race te simuleren dus ook falende racemiddelen. Dit ga je in deze taak toevoegen aan je simulator.

Aanpak

Tijdens het implementeren van het klassendiagram heb je al de property `IsBroken` aangemaakt bij de interface `IEquipment`. Deze property ga je in deze taak gebruiken. Bepaal vooraf hoe je de kans gaat bepalen dat een racemiddel kapot gaat. Hiervoor kan je de andere properties voor gebruiken van `IEquipment`.

Omdat de klasse `Race` de simulatie voor zijn rekening neemt ga je deze klasse uitbreiden en aanpassen.

- Breid de klasse `Race` uit waarbij je bij elke `Elapsed` event de property `IsBroken` af en toe op `true` zet. Gebruik hiervoor de gekozen kansberekening en de randomizer.

  Wanneer je de simulatie weer aanzet zie je op de console nog weinig gebeuren. Bedenk een leuke character om visueel te laten zien dat een deelnemer pech heeft.

- Pas de visualisatie aan waarbij je zichtbaar maakt wanneer `IsBroken` property op `true` staat voor een deelnemer.

  Nu je het racemiddel van de deelnemer stuk hebt gemaakt kan het natuurlijk niet zo zijn dat de deelnemer toch verder mag.

- Pas de klasse `Race` aan en zorg ervoor dat deelnemers niet bewegen wanneer de `IsBroken` property op `true` staat.

  Met wat geluk krijg je een race te zien die nooit eindigt omdat alle deelnemers pech hebben. Toch is het de bedoeling dat alle deelnemers de race weten af te ronden. Bepaal nu hoeveel kans een deelnemer heeft om weer verder te kunnen met de race.

- Pas de klasse `Race` nogmaals aan en bepaal voor alle deelnemers, waarbij de property `IsBroken` op `true` staat, of de value weer op `false` gezet mag worden. Gebruik hiervoor weer de randomizer.

  Het is fijn dat deelnemers weer de race kunnen afronden maar schade aan je racemiddel kan uiteraard niet zonder gevolgen zijn.

- Voor elke deelnemer, waarbij de property `IsBroken` op false wordt gezet, pas tenminste 1 property aan die invloed heeft op snelheid en/of betrouwbaarheid van het racemiddel.

  Nu is de race simulator weer een stukje realistischer. Deelnemers die bij aanvang van de race erg snel zijn kunnen nu toch als laatste eindigen.