

# spASDinvaders

Bijdrage van Hans Kuppens aan de ASD:Suite Contest 2011

**HANS KUPPENS**

10 juli 2011

# spASDInvaders

Bijdrage van Hans Kuppens aan de ASD:Suite Contest 2011

## Inleiding

spASDInvaders is een illustratieve demonstratie van de mogelijkheden die ASD:Suite biedt. spASDInvaders is geïnspireerd op het oude computerspel "SpaceInvaders", een arcade spel uit 1978, zie ook [http://nl.wikipedia.org/wiki/Space\\_Invaders](http://nl.wikipedia.org/wiki/Space_Invaders).

Voor de latere generaties onder ons: het is de bedoeling om met een laserkanon (uit 1978!) een eskader van ruimtewezens te vernietigen, die in zigzagbeweging naar beneden komen en terugschieten. Naarmate er meer ruimtewezens vernietigd zijn, worden de bewegingen van de overgebleven wezens steeds sneller. Soms komt er een ruimteschip voorbij aan de bovenkant, dat kapot geschoten kan worden voor extra punten. Als alle wezens zijn vernietigd heeft de speler gewonnen, echter als het laserkanon geraakt wordt of als de wezens onder een bepaalde hoogte gedaald zijn heeft de speler verloren.

In het oorspronkelijke spel heeft het laserkanon 3 levens, en komen de ruimtewezens in een nieuw scherm terug als het laatste wezen vernietigd is; deze functies zijn echter (nog) niet in deze demonstratie geïmplementeerd. Ook de geluidseffecten ontbreken in de demonstratie, evenals een "highscore" functie. Desondanks zijn met deze demonstratie de mogelijkheden van ASD:Suite in ruime mate geïllustreerd.

## Ontwikkelomgeving

De applicatie is ontwikkeld met ASD:Suite Release 3 v7.x.x (uiteraard!), en voor de rest met MicroSoft Visual C# 2010 Express.

De sourcefiles zijn ondergebracht in de volgende directories:

spASDInvaders	hoofddirectory met generieke project files
spASDInvaders\AsdModels	alle ASD modellen van het project (*.im en *.dm)
spASDInvaders\AsdComponents	alle gegenereerde code vanuit de ASD modellen (do not edit!)
spASDInvaders\ForeignComponents	alle handgeschreven code met ASD interfaces
spASDInvaders\AsdRuntime	de ASD runtime bestanden, inclusief ITimer.im
spASDInvaders\Resources	alle grafische objecten, in de vorm van PNG bitmaps
spASDInvaders\Properties	overige generieke project files

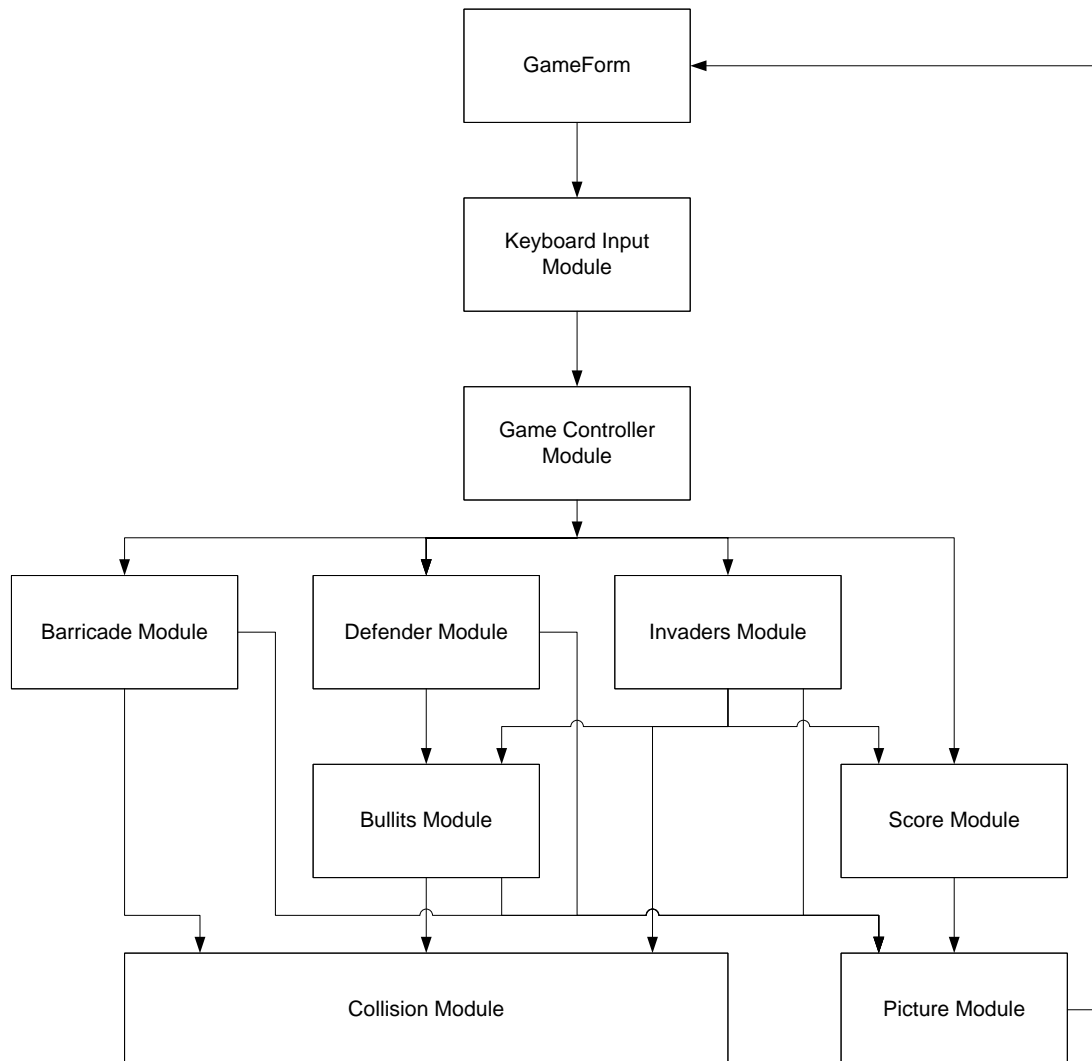
Overige directories:

spASDInvaders\bin	gegenereerde binaire bestanden
spASDInvaders\obj	gegenereerde binaire bestanden
spASDInvaders\publish	gegenereerde distributie bestanden

Om spASDInvaders op een computer te installeren voer je het programma spASDInvaders\publish\setup.exe uit.

## Architectuur

In onderstaand diagram is de architectuur van de applicatie schematisch weergegeven.



De modules worden hieronder toegelicht; voor het gedetailleerde diagram met de actuele componenten wordt naar het bijgevoegde spASDinvader.vsd Visio-document verwezen.

### GameForm Module

Vanuit het applicatie scherm “GameForm” worden keyboard events naar de Keyboard module gestuurd. Tijdens GameForm\_Load() worden de ASD componenten geïnstantieerd en geïnitialiseerd. Omgekeerd, tijdens GameForm\_FormClosing() worden de ASD componenten getermineerd en verwijderd; dit is niet toegestaan tijdens het spelen van het spel.

Via de Picture module worden grafische objecten op het GameForm weergegeven, door dynamisch objecten van het type PictureBoxPlus (afgeleid van het System.Drawing PictureBox object) te creëren. Het type PictureBoxPlus is achteraf allicht niet nodig geweest, maar is ontstaan om ‘delegates’ te implementeren: functies die uitgevoerd worden in de thread van de user interface.

## Keyboard Module

Hierin worden keyboard events volgens het postman-pattern afgevangen. Hiermee wordt de thread van de user interface zo snel mogelijk weer vrijgegeven, en de keyboard events op de DPC thread verder afgehandeld.

De postman is geïmplementeerd in KeyboardControllerDataComponent.cs, die intern een event queue bevat. Tussen de postman en KeyboardController.dm is een protocol dat de events een voor een toestaat. Zonder dit protocol wordt KeyboardController overladen met events, waardoor de ModelCheck tijd drastisch toeneemt. Een alternatief voor dit protocol is het yoking mechanisme, dat elders (in de CollisionDetector) wordt toegepast.

De keyboard module doet verder wat elementaire filtering, o.a. om de 'left' en 'right' key events om te zetten in joystick commando's. Ook worden repeterende key events weggenomen.

Tenslotte heeft de functie CanTerminate() enige toelichting. Op het moment dat het spel in volle gang is, worden vanuit de vele DPC threads (als gevolg van timer events) updates op de UI uitgevoerd (via de Picture module). Op het moment dat een GameForm\_FormClosing() optreedt, kan deze niet alle componenten termineren want de UI update is dan tijdelijk geblokkeerd, en dus ook de componenten waarvan de DPC de UI update in uitvoering heeft. Deze deadlock situatie is voorkomen door tijdens het spel geen Terminate() aan te roepen, de functie CanTerminate() geeft dat aan.

## GameController Module

Deze bepaalt het spelverloop. Initieel wordt het "welkomstschermb" getoond, en staan alle used components op "Initialized". Na het keyboard event 'S' (start) worden alle used components met StartPlay() op "Playing" gezet. Het spel kan eindigen op de volgende manieren:

- IInvaderSquadronCB.OnDestroyed() → Spel gewonnen
- IInvaderSquadronCB.OnTouchedDown() → Spel verloren
- IDefenderCB.OnDestroyed() → Spel verloren
- IGameCtrl.Start() → Spel voortijdig beëindigd (functie Start/Stop combi)

Als het spel is geëindigd, wordt het slagveld opgeruimd, en het van toepassing zijnde 'uitslag' scherm getoond.

## Picture Module

Deze module bestaat uit een handgeschreven 'multiple' component PictureAccessorComponent.cs, waarmee alle visuele elementen gerealiseerd kunnen worden. Bitmaps worden zichtbaar of verborgen gemaakt, naar een positie bewogen, afmetingen opgevraagd, enzovoort. De Picture module is volledig multithreaded, de actie op het GameForm vindt altijd in de UI thread plaats door het gebruik van de eerder genoemde 'delegates'.

## Barricade Module

De barricade module bestaat uit 4 groepjes van 10 "BarricadeBricks" elk. Deze BarricadeBricks zijn geïmplementeerd als een UCV van 40 elementen. Via de singleton BarricadeData component krijgt elke steen een eigen positie op het slagveld.

Elke steen kan 4x door een "Bullit" geraakt worden voordat deze vernietigd is, m.u.v. de schuine stenen die na 2x al weg zijn. Na elke inslag verandert het patroon van de steen. Als alle stenen vernietigd zijn, hoeft dit niet via een callback doorgegeven te worden aan de GameController – voor het verdere verloop van het spel maakt dit immers niet uit. Om deze reden geeft elke BarricadeBrick ook geen callback aan de BarricadeLine door dat deze vernietigd is.

## Defender Module

Het laserkanon wordt met MoveLeft(), MoveRight() en MoveStop() commando's bestuurd, en schiet met Shoot() een "Bullit" af. Pas als deze Bullit een doel geraakt heeft, of boven van het scherm verdwenen is, kan een nieuwe Bullit worden afgevuurd. Als de Defender zelf door een Bullit geraakt wordt, wordt de callback OnDestroyed() aangeroepen.

## Invaders Module

In totaal zijn er 5 rijen van elk 9 "Invaders". Elke rij is verdeeld in 3 groepjes van 3 Invaders, zie verderop voor een toelichting.

Dus: InvaderSquadron.dm heeft een UCV van InvaderLine[5], InvaderLine.dm een UCV van InvaderGroup[3], InvaderGroup een UCV van Invader[3].

Het versnellen van de ruimtewezens (naarmate er meer vernietigd zijn) ontstaat doordat InvaderSquadron in een vast ritme een MoveXXX() commando geeft, die door telkens één van de overgebleven wezens wordt uitgevoerd. XXX staat dan voor de richting: Right, Left of Down. Dus hoe minder overgebleven wezens, hoe sneller de beweging.

Voorafgaand aan elke MoveXXX() worden eerst alle overgebleven wezens gevraagd of een move in die richting mogelijk is. Zo ja, dan wordt de MoveXXX() herhaald aangeroepen totdat de response retNoMore terugkomt, waarna de vraag herhaald wordt. Kan een move niet meer plaatsvinden, dan verandert de richting (Right-Down-Left-Down-Right-Down-...). Als uiteindelijk Down niet meer mogelijk is dan is het spel afgelopen, en heeft de speler verloren. Dit wordt middels de callback OnTouchedDown() kenbaar gemaakt.

Interessant in het ontwerp is de omgang met UCV state variabelen. Immers, bij MoveLeft() is de volgorde waarin Invaders worden aangeroepen omgekeerd vergeleken bij MoveRight(). Bovendien, Invaders die vernietigd zijn komen niet meer in de UCV state variabelen voor.

Elke Invader geeft een callback OnDestroyed() als deze door een Bullit vernietigd is. Een InvaderGroup geeft een callback OnDestroyed() als alle Invaders van het groepje vernietigd zijn, enzovoort. Als alle 5 InvaderLine's een OnDestroyed() hebben gegeven, geeft InvaderSquadron ook een OnDestroyed() door en is het spel afgelopen, en heeft de speler gewonnen.

Nog een toelichting op het waarom er geen UCV van Invader[45] gekozen is. Elke Invader kan een 'unsolicited' callback OnDestroyed() geven op elk willekeurig moment. Een korte test heeft mij geleerd dat dit tot excessieve ModelCheck leidt. In het experiment bleek een UCV van 5 nog goed te doen, 6 was maximaal haalbaar. Later heb ik van Leon Bouwmeester begrepen dat het voor dit soort situaties mogelijk is om in de ModelCheck een kleiner aantal UCV's te gebruiken dan in de feitelijke code generatie. Nu heeft dit tot oneigenlijke tussen-componenten geleid. Bij de BarricadeBrick speelt dit overigens niet, vanwege het ontbreken van unsolicited events daar.

Er is ook nog de InvaderShip.dm component, die rechtstreeks onder de GameController hangt. Had ook onder InvaderSquadron kunnen hangen, maar die had al 5x InvaderLine (met callback), dus deze component wilde ik sparen. Had ook een InvaderArmy component kunnen maken die InvaderSquadron en InvaderShip combineerden (was goed voor de ASD contest punten ;-)) geweest maar toch niet gedaan).

Het InvaderShip vertrekt op willekeurige momenten, en geeft een random score indien vernietigd (100, 150, 200 of 250 punten). Het is niet nodig om vernietiging middels een OnDestroyed() callback door te geven.

## Bullits Module

Elke Invader heeft één "Bullit" die met een timer event wordt afgeschoten; de Defender heeft uiteraard ook een Bullit, die wordt afgevuurd met het Launch() commando. Pas als de Bullit een doel heeft geraakt, of uit het

slagveld is verdwenen, kan een nieuwe Bullit worden afgevuurd; beide situaties worden met een `OnDestroyed()` callback doorgegeven aan de betreffende “eigenaar”.

## Collision Module

In de collision module vindt de bewaking plaats of een Bullit in aanraking komt met een `DestroyableObject`.

Het hart van de collision module is de `CollisionDetectorComponent.cs`, een handgeschreven component met twee interface channels en een broadcast channel.

Via de interface `ICollision` geven de `DestroyableObjects` (de `BarricadeBricks`, de `Invaders`, de `InvaderShip` en – last but not least – de `Defender`) hun posities en afmetingen door. Deze worden in interne lijsten bijgehouden, samen met een soort “in leven” vlaggetje.

Via de interface `ICollisionBullit` geeft elke Bullit zijn bewegingen door. Bij elke beweging wordt door de lijsten van `DestroyableObjects` gebladerd of de nieuwe bullit positie binnen de rechthoek van een van de objecten valt. Zo ja, dan wordt dat synchroon aan de Bullit component teruggemeld, en wordt tevens een event op het broadcast channel geplaatst. De Bullit weet nu dat een doel is geraakt, en zal zichzelf dus vernietigen.

En als de Bullit geen doel raakt, kan deze zijn weg vervolgen en komt er ook geen event op het broadcast channel.

Het broadcast channel heeft de volgende events:

```
void OnBrickCollision(int brickId);
void OnDefenderCollision();
void OnInvaderCollision(int invaderId);
void OnInvaderShipCollision();
```

De collision module houdt er rekening mee dat een `Invader` niet geraakt kan worden door een Bullit van een andere `Invader`.

In de interface van de `CollisionDetector` wordt geen belofte over de state van de `DestroyableObjects` gedaan. Een `OnXxxCollision()` event kan dus te allen tijde komen. Voor de `Bricks` en `Invaders` kan er zelfs geen state belofte gemodelleerd worden, want deze onderscheiden zich onderling slechts met een parameter dat het ID ervan aanduidt. Als oplossing zit tussen elk `DestroyableObject` en de `CollisionDetector` een proxy (die geen singleton is). Deze proxy houdt wél de state (en zondig het ID) van het `DestroyableObject` bij, en filtert de overtollige `OnXxxCollision()` events keurig weg.

Om de modelchecks van deze proxies binnen de perken te houden, worden de events met yoking begrensd. Een protocol zoals bij de keyboard events is hier immers niet mogelijk tussen de vele proxies en de singleton `CollisionDetector`.

## Score Module

Tenslotte is er de score module, waarin o.a. een UCV van 4 digits. Via de `IScoreCtrl` interface kan de teller worden gestart en gestopt, terwijl via de `IScoreAdd` interface de puntenteller kan worden opgehoogd. Dit laatste gebeurt vanuit elke `Invader` component, en natuurlijk ook vanuit de `InvaderShip` component met de ‘mystery’ puntenwaardering.

## Nawoord

**Het was mij een genot om aan deze ASD:Suite Contest mee te hebben gedaan.**