Game Maker Tutorial

Multiplayer Spellen Maken

Geschreven door Mark Overmars

Vertaling: Joris Vergeer, Maarten van de Peppel & Jelle Straatsma
Nederlandse Game Maker Community (www.game-maker.nl)
Laatst veranderd: 25 februari 2007
Gebruikt: Game Maker 7.0, Pro Editie, Advanced Mode
Niveau: Gevorderd

Het spelen van games tegen de computer is leuk. Maar games spelen tegen andere mensen kan nog veel leuker zijn. Het is over het algemeen veel makkelijker om te maken omdat je geen ingewikkelde computer ai hoeft te maken. Je kunt natuurlijk een spel maken met 2 spelers achter dezelfde computer met andere toetsen of invoer apparaat(zoals joystick, muis, toetsenbord, etc.) maar het interessanter als elke speler achter een eigen computer kan zitten. Of nog beter, spelen tegen iemand aan de andere kant van de wereld, met *Game Maker* is dat mogelijk. In deze tutorial wordt uitgelegd hoe je het moet gebruiken. Maar wees gewaarschuwd, het maken van een een dergelijk multiplayer game is niet gemakkelijk. Her vereist een ervaren *Game Maker* gebruiker. Je zult sommige codes moeten begrijpen. Dus laat dit niet het eerste spel zijn dat je maakt. Om dit te kunnen gebruiken heb je een geregistreerde versie van *Game Maker* nodig.

In deze tutorial gaan we een simpel pong spel voor twee spelers maken en een klein chat programma. Alle spellen zijn te vinden in de map `Examples` en kunnen geladen worden in *Game Maker*. Het gaat hier niet over het uiterlijk maar over het multiplayer aspect. Je kunt de basis gebruiken om mooiere spellen te maken. We zullen de volgende punten behandelen:

- Verbinding maken met aan andere computer
- Het maken of deelnemen van een sessie
- Het spel gesynchroniseerd houden

Dit laatste is het moeilijkst in alle multiplayer spellen. Het probleem is zeker te zijn dat alle spelers hetzelfde zien. Bijvoorbeeld, in het pong spel, beide spelers moeten de bal op dezelfde plaats zien. *Game Maker* heeft een hulpmiddel om dit te doen maar je moet zelf de connectie ontwerpen voor elk spel die je maakt.

Verbinding maken

Normaal werkt een multiplayer spel als volgt. Elke speler opent hetzelfde spel. Maar spelen het in verschillende modes. Een speler speelt het spel in server mode. De ander in cliënt mode. De server start het spel het eerst en maakt een spel sessie. De ander kan dan de sessie joinen (join is Engels voor meedoen). De spelers moeten kiezen welke methode ze gebruiken om de computers met elkaar te laten communiceren. Op een lokaal netwerk is het het makkelijkst om een IPX connectie te gebruiken (beneden meer details). Als alle spelers verbinding maken over internet wordt meestal TCP/IP gebruikt. Met dit protocol moeten cliënten het IP adres van de server weten. Dus de

speler die het spel in server mode speelt moet dus zijn IP adres aan de andere spelers geven (bijvoorbeeld over e-mail). Je kunt achter je IP adres komen door het programma `winipcfg.exe` te draaien dat in je Windows map staat. Je kunt ook achter je IP adres komen met de *Game Maker* functie `mplay_ipaddress()` . Een wat oudere manier is een modemverbinding (waarbij de cliënt het telefoonnummer van de server moet weten) of een seriële kabel.

Waarschuwing: Denk er wel aan dat het tegenwoordig moeilijker is omdat de meeste mensen tegenwoordig een firewall en/of router hebben. Deze blokkeren berichten en veranderen het IP adres. Als je problemen hebt met een verbinding tot stand te brengen is dus mogelijk de reden. Het best is om eerst te testen met een bestaan spel of je problemen hebt met de verbinding. Zie de volgende site voor informatie over hoe je met dit probleem om kunt gaan:

http://support.microsoft.com/default.aspx?scid=http://support.microsoft.com:80/support/kb/articles/Q240/4/29.ASP&NoWebContent=1

Denk eraan dat het interne IP adres niet altijd hetzelfde is als het externe IP adres dat door een router veroorzaakt kan worden.

Als twee computers met elkaar hebben ze een verbindingsprotocol nodig. Net als de meeste spellen heeft *Game Maker* vier verschillende soorten verbindingen: IPX, TCP/IP, modem en seriël. De IPX verbinding werkt bijna helemaal onzichtbaar. Het kan gebruikt worden om spellen te spelen over een lokaal netwerk. Het protocol moet geïnstalleerd worden op de computer als je het wilt gebruiken. Je kunt het installeren in netwerk in het configuratiescherm. TCP/IP is het internet protocol. Het kan gebruikt worden om te spleen met mensen overall op het internet, als de spellers elkaars IP maar weet. Op een lokaal netwerk hoef je geen IP adres op te geven. Een modemverbinding maak je met een modem. Als je deze verbinding gebruikt moet je een paar instellingen invoeren (een initialisatie string en telefoonnummer) om het te gebruiken. En ten slotte een seriële kabel verbinding (een directe verbinding tussen twee computers). Om dit te gebruiken moet je wat poort instellingen ingeven. Er zijn vier GML functies die je kunt gebruiken om een verbinding te initialiseren:

- `mplay_init_ipx()` initaliseert de IPX verbinding.
- `mplay_init_tcpip(addr)` initaliseert de TCP/IP verbinding. `addr` is een string die een IP adres of internet adres bijv. ‘www.gameplay.com’ of ‘123.123.123.12’, mogelijk gevolgd door een poort nummer (bijv. ‘:12’). Alleen als je deelneemt (join) aan een sessie (later meer) moet de een adres invoeren. De persoon die een sessie maakt hoeft geen adres in te vullen. Op een lokaal netwerk(LAN) hoef je geen adres in te voeren.
- `mplay_init_modem(initstr,phonenr)` initaliseert de modemverbinding. `initstr` is de initalisatiestring voor het modem (kun ja leeg laten). `phonenr` is het telefoonnummer dat gebeld moet worden (bijv. 0201234567). Alleen als je een sessie ‘joint’ (zie beneden) moet je het telefoonnummer invoeren.
- `mplay_init_serial(portno,baudrate,stopbits,parity,flow)` initaliseert een seriële verbinding. `portno` is het poortnummer (1-4) baudrate is het baudrate dat gebruikt moet worden (100-256K). `stopbits` bepaalt het

aantal stopbits dat gebruikt moet worden (0 = 1 bit, 1 = 1.5 bit, 2 = 2 bits). `parity` bepaalt even of oneven (0=geen, 1=oneven, 2=even, 3=mark). En `flow` stelt het type flow controle in (0=geen, 1=xon/xoff, 2=rts, 3=dtr, 4=rts en dtr). Geeft terug of het succesvol was. Een kenmerkende invulling is: `mplay_init_serial(1,57600,0,0,4)`. Schrijf 0 als eerste argument om een dialoog met de speler te openen om deze instellingen te veranderen.

Je spel moet één van deze functies een keer gebruiken. Alle functies geven terug als het succesvol was. Ze zijn onsuccesvol als het gebruikte protocol niet beschikbaar is of als deze niet ondersteund wordt door de computer.

In de eerste room in je spel zou je de speler moeten laten kiezen tussen deze vier verbindingen. (of zet alleen de protocollen neer die je wilt gebruiken) We roepen de initialisatie functies op in het muis event en, als succesvol, gaan we naar de volgende room. Anders komt er een fout bericht. Dus in het muis event zetten we het volgende stukje code:

```
{
  if (mplay_init_ipx())
    room_goto_next()
  else
    show_message('Initialiseren van een IPX verbinding is mislukt.')
}
```

Als het spel stopt of als je spel geen multiplayer meer nodig hebt kun je de volgende code gebruiken:

- `mplay_end()` maakt een eind aan de momentele verbinding. Geeft terug of het succesvol is.

Dit moet je gebruiken als je een nieuwe andere verbinding wilt maken.

Spel sessies

Als je verbinding hebt met een netwerk, kunnen daar meerdere spellen bezig zijn. Deze spellen noemen we sessies. Deze verschillende sessies kunnen van hetzelfde spel of van verschillende spellen zijn. Een spel moet zichzelf uniek kunnen identificeren. Gelukkig doet *Game Maker* dit voor je. Het enige dat je moet weten is dat als je de game id in het opties scherm veranderd, ook de identificatie word veranderd. Op deze manier kun je voorkomen dat mensen met oude versies tegen mensen met nieuwe versies gaan spelen.

Als je een nieuw multiplayer spel start, moet je een sessie maken. Hiervoor gebruik je de volgende functie:

- `mplay_session_create(sesname,playnumb,playername)` maakt een nieuwe sessie op de huidige verbinding. `sesname` is een regel die de naam van de sessie bevat. `playnumb` is het maximale aantal spelers (gebruik 0 voor onbeperkt). `playername` is je naam als speler. Geeft terug of het succesvol was.

In veel gevallen is worden spelernamen niet gebruikt dus kan de string leeg gelaten worden. De sessienaam is alleen nodig als je spellers de keuze willen geven om te kiezen tussen sessies.

Een speler moet de sessie maken en al de andere spelers van het spel kunnen meedoen aan de sessie (joinen). Dit is iets moeilijker. Je moet eerst kijken welke sessies er beschikbaar zijn, en er dan één kiezen om te joinen. Er zijn hiervoor drie belangrijke functies:

- `mplay_session_find()` zoekt alle sessies die nog steeds spelers accepteren en geeft het aantal sessies terug.
- `mplay_session_name(numb)` geeft de naam terug van het sessie nummer opgeslagen in 'numb' (0 is de eerste sessie). Deze functie kan alleen gebruikt worden nadat de vorige functie is gebruikt.
- `mplay_session_join(numb,playername)` laat je de sessie `numb` joinen (0 is de eerste sessie). `playername` is je naam als speler. Geeft terug of het succesvol was.

Normaal gebruik de functie `mplay_session_find()` om beschikbare sessie te zoeken. Daarna gebruik je herhaaldelijk `mplay_session_name()`om de speller beschikbare sessie te tonen of je laat automatisch de eerste sessie joinen. (Merk op dat het veel tijd kost om beschikbare sessies te zoeken, dus gebruik deze functie niet elke stap.)

Een speler kan stoppen met de huidige sessie met de volgende functie:

- `mplay_session_end()` beëindigt de sessie voor deze speler.

Het kan nuttig zijn om dit de andere spelers te vertellen maar het is niet noodzakelijk.

De spellen van deze tutorial geven in de tweede room de speler twee keuzes: een sessie maken of een sessie joinen. Om een sessie te maken hebben we de volgende code in het muis event gezet:

```
{
  if (mplay_session_create('',2,'')){
    global.master = true;
    room_goto_next();
  }
  else
    show_message('Kan geen sessie maken.')
}
```

Belangrijk is dat we een globale variabele op true zetten. De reden daarvoor is dat we onderscheid moeten maken tussen de eerste speler (de master) en de tweede speler (de slave). De master is verantwoordelijk voor het grootste deel van de game play, de slave volgt de master gewoon.

De tweede keuze is het meedoen aan een bestaand spel. Daarvoor gebruiken we de volgende code:

```
{
  if (mplay_session_find() > 0){
    if (mplay_session_join(0,'')){
      global.master = false;
      room_goto_next();
    }
    else
      show_message('Kan niet worden toegevoegd aan sessie.')
    }
  else
    show_message('Geen sessie om aan toe te worden gevoegd.')
}
```

Met deze code wordt je simpel toegevoegd aan de eerste sessie. Omdat er maar twee spelers kunnen spelen kan niemand anders meer meedoen.

Omgaan met spelers

Zodra een master een sessie heft gemaakt, moeten we alleen nog wachten op andere spellers. Er zijn drie functies die met spelers te maken hebben.

- `mplay_player_find()` zoekt het aantal spelers en geeft het aantal gevonden spellers terug.
- `mplay_player_name(numb)` geeft de naam van de speler `numb` terug (0 is de eerte speller en dat bij jij altijd zelf.). Deze functie kan alleen gebruik worden na de vorige functie.
- `mplay_player_id(numb)` geeft het unieke id van een speler `numb` terug (0 is de eerste speler, die je altijd zelf bent). Deze functie kan alleen gebruikt worden na de eerste functie. Dit id wordt gebruikt om berichten te zenden en ontvangen tussen individuele spelers.

In onze derde room wachten we gewoon op een tweede speler om mee te doen. Om dat te doen maken we een object en zetten in het step event:

```
{
  if (mplay_player_find() > 1)
    room_goto_next();
}
```

(Dit is natuurlijk niet nodig voor de speler die de sessie joint.)

Acties synchroniseren

Nu hebben we een verbinding gemaakt, een sessie gemaakt en we hebben twee spelers. Het echte spel kan beginnen. Maar dit betekend ook dat je moet beginnen met nadenken over de verbinding. Het grootste probleem in een multiplayer spel is de synchronisatie. Hoe zijn we er zeker van dat beide spelers precies hetzelfde zien? Dit wel heel belangrijk. Waneer beide spelers de bal op een andere plaats zien, kunnen er vreemde dingen gebeuren. (In het ergste geval ziet de ene speler dat de bal gemist wordt terwijl de ander ziet dat hij de bal wel raakt.) Het spel raakt dan uit ballans en dat zorgt voor een grote chaos in het spel.

Wat erger is is dat we dit soms met minder berichten moeten doen. Als je bijvoorbeeld speelt over een modem dan is de verbinding traag. En je wilt natuurlijk

zoveel berichten overzenden als mogelijk is. Ook kan er een vertraging zitten tussen het verzenden en ontvangen van berichten. Het is zelf mogelijk dat gegevens niet aankomen.

Hoe je deze problemen oplost hangt af van wat voor soort spel je maakt. In een op beurten gebaseerd spel zit waarschijnlijker een simpeler systeem dan in een snelle actie spellen.

Game Maker kent twee manieren om te communiceren met elkaar: data delen en berichten. Data delen is de simpelste manier. Berichten verzenden is moeilijker maar je kunt er wel meer mee.

Data delen

Het delen van data is waarschijnlijk de gemakkelijkste manier om je spel te synchroniseren. Alle communicatie is van je afgeschermd. Er is een set van 1000000 waardes die elke speler in het spel kan herkennen (liever alleen de eerste paar waardes om geheugen te besparen). Elke speler kan waardes schrijven én lezen. *Game Maker* zorgt ervoor dat elke speler de juiste waardes ziet. Een waarde kan een regel of een nummer zijn. Er zijn slechts twee functies:

- `mplay_data_write(ind,val)` schrijft de waarde `val` (tekenreeks of getal waarde) in nummer `ind` (`ind` tussen 0 en 1000000).
- `mplay_data_read(ind)` geeft de waarde terug die zich in `ind` bevindt (`ind` tussen 0 en 1000000). Bij het starten van het spel zijn alle waardes 0.

Om dit te gebruiken moet je van te voren bepalen wat welke waardes zijn en wie bepaalde waardes kan veranderen. Het is aangeraden om een object te hebben die de waardes schrijft. Wat ook aangeraden is is om alleen de eerste paar indicatoren (bij de bovengenoemde twee functies de variabele ind) te gebruiken om geheugen te sparen.

In ons geval zijn er vier belangrijke waardes: de y-positie van de master, de y-positie van de slave en de x en y positie van de bal. Wij gebruiken indicator 1 voor de y-coordinaten van de master, indicator 2 voor de y-coordinaten van de slave, etc. Het is duidelijk dat de master zijn waardes schrijft en de slave de zijne. We hebben besloten dat de master verantwoordelijk is voor de waardes van de bal. De slave hoeft alleen elke step de bal op de juiste posities te drawen.

Hoe laten we dit werken? Eerst, het master bat (de linkse) kan alleen door de master bestuurd worden. Dit betekend dat de omhoog en omlaag toetsenbord event die we daarvoor gebruiken alleen mag werken bij de master. De code die we daar voor gebruiken is iets als:

```
{
  if (!global.master) exit;
    if (y > 104) y -= 6;
      mplay_data_write(1,y);
}
```

Voor de slave gebruiken we een vergelijkbaar stukje code om de bat van de master op de goede plek te zetten. Dit doen we elke step met de volgende code:

```
{
  if (global.master) exit;
    y = mplay_data_read(1);
}
```

Hetzelfde voor de bat van de slave.

Ten slotte, eerst moeten we ervoor zorgen dat de bal stuitert tegen de muren en de bats. Wij zorgen ervoor dat alleen de master dit doet. Om de positie van de bal naar de slave te sturen zetten we de volgende code in het step event:

```
{
  if (global.master){
    mplay_data_write(3,x);
    mplay_data_write(4,y);
  }
  else
  {
    x = mplay_data_read(3);
    y = mplay_data_read(4);
  }
}
```

Hiermee is de basis klaar. Wat over blijft is het regelen van het begin van het spel, het bijhouden van de scores, etc. Dit wordt allemaal door de master geregeld. Dus de master controleert waneer een speler verliest, veranderd de score, en zorgt dat de bal op de goede plek staat. De details kun je zien in het bestandje: `pong1.gmk`.

Merk wel op dat zoals het communicatie schema uitlegt de twee spellen een beetje ongesynchroniseerd kunnen lopen. Normaal is dit geen probleem. Je kunt dit oplossen door een synchronisatie object te maken die controleert of alles wel goed is voordat er iets getekend wordt. Dit moet wel met enige zorg gemaakt worden anders krijg je problemen.

Denk eraan dat als er een speler bijkomt de al veranderde variabelen NIET naar de nieuwe speler worden verzonden. Allen de veranderingen nadat een speler zich heeft aangemeld worden naar de nieuwe speler verzonden.

Berichten

Het tweede communicatie mechanisme dat *Game Maker* ondersteund is het sturen en ontvangen van berichten. Een speler kan berichten sturen naar één of alle andere spelers. Spelers kunnen zien of berichten zijn aangekomen en actie ondernemen aan de hand van dat bericht. Berichten kunnen worden verstuurd in gegarandeerde modus of in een niet gegarandeerde modus, wat sneller is.

We zullen eerst berichten gaan gebruiken om een paar geluiden toe te voegen aan ons spel. We hebben een geluidje als de bal de bat raakt, als de bal de muur raakt en als een speler wint. Alleen de master kan controleren of een van deze dingen gebeurt. Dus de master moet beslissen wanneer er een geluidje bij de slave wordt afgespeeld. Het is gemakkelijk te controleren op één computer. Je hoeft dan alleen het geluidje af te spelen. Maar hoe vertellen we de slave dat hij een geluidje moet afspelen. We

kunnen hier “gedeelde data” voor gebruiken maar dat is lastig. Berichten gebruiken is makkelijker. De master hoeft alleen een berichtje naar de slave te sturen dat het geluidje afgespeeld moet worden. De slave hoeft alleen te luisteren naar het bericht en het goede geluidje al te spelen.

De volgende functies voor berichten bestaan in *Game Maker*:

- `mplay_message_send(player,id,val)` stuurt een bericht naar de aangegeven `player` (gebruik of de naam of het id van een speler; gebruik 0 om het bericht aan alle spelers te sturen). `id` is een geheel getal dat het id van het bericht aangeeft en `val` is de waarde in het bericht (een getal of een regel). Het bericht wordt verzonden in een niet gegarandeerde modus.
- `mplay_message_send_guaranteed(player,id,val)` stuurt een bericht naar de aangegeven `player` (gebruik of de naam of het id van een speler; gebruik 0 om het bericht aan alle spelers te sturen). `id` is een geheel getal dat het id van het bericht aangeeft en `val` is de waarde in het bericht (of een getal of een regel). Het bericht wordt verzonden in een gegarandeerde modus.
- `mplay_message_receive(player)` ontvangt het volgende bericht uit de wacht rij van de aangegeven `player` (gebruik de naam of het id van een speler). Gebruik 0 om berichten van iedere speler te ontvangen. De functie geeft terug of er inderdaad een nieuw bericht was. Als dat het geval is kun je de volgende functies gebruiken om de inhoud van de berichten te krijgen:
- `mplay_message_id()` Geeft het id van het laatst ontvangen bericht terug.
- `mplay_message_value()` Geeft de waarde van het laatst ontvangen bericht terug.
- `mplay_message_player()` Geeft de speler terug die het laatst ontvangen bericht stuurde.
- `mplay_message_name()` Geeft de naam van de speler terug die het laatst ontvangen bericht stuurde.
- `mplay_message_count(player)` Geeft het aantal berichten terug dat nog steeds in de wacht rij staat van een bepaalde `player` (gebruik 0 om alle berichten te tellen).

Een paar opmerkingen zijn hier op zijn plaats. Ten eerste, als je een bericht wilt verzenden naar één bepaalde speler, moet je zijn unieke id weten. Zoals eerder aangegeven kun je dit te weten komen met de functie: `mplay_player_id()`. Deze speler id word ook gebruikt om berichten van één bepaalde speler te ontvangen. Een alternatieve manier om dit te doen is de naam van de speler geven als tekenreeks. Als meerdere spelers dezelfde naam hebben krijgt alleen de eerste het bericht.

Ten tweede vraag je je misschien af waarom ieder bericht een eigen id heeft. De reden hiervoor is dat het je spel helpt om verschillende typen berichten te sturen. De ontvanger kan het type bericht checken en passende actie ondernemen. (Omdat berichten niet gegarandeerd aankomen zou het sturen van id en waarde in verschillende berichten serieuze problemen opleveren.)

Voor het afspelen van geluiden doen we het volgende. De volgende code wordt uigevoerd als de master ontdekt dat de bal de bat raakt.

```
{
  if (!global.master) exit;
    sound_play(sound_bat);              // speel het geluid zelf af
    mplay_message_send(0,100,sound_bat); // verstuur naar de ander
}
```

Het controller object doet in het step event het volgende:

```
{
  while (mplay_message_receive(0))
  {
    if (mplay_message_id() == 100)
      sound_play(mplay_message_value());
  }
}
```

Dit controleert of er een bericht binnenkomt en welk id het heeft. Als het id 100 is wordt het geluid afgespeeld die het bericht bevat.

Nu weer even in het algemeen, je spel heeft nu een typisch controller object die in het step even iets doet zoals:

```
{
  var from, name, messid, val;
  while (mplay_message_receive(0)
  {
    from = mplay_message_player();
    name = mplay_message_name();
    messid = mplay_msesage_id();
    val = mplay_message_value();
    if (messid == 1)
    {
      // doe iets
    }
    else if (messid == 2)
    {
      // doe iets anders
    }
    // etc.
  }
}
```

Het ontwerpen van het communicatieprotocol (dat is, bepalen welke berichten zijn verzonden, door wie en op welke momenten, en hoe de andere spelers daarop moeten reageren) is zeer belangrijk. Ervaring en kijken naar voorbeelden van anderen helpt veel.

Dead-reckoning

Het pong spel heeft een serieus probleem. Als er iets fout gaat met de verbinding staat de bal bij de slave tijdelijk stil. Als er geen nieuwe coördinaten binnenkomen, beweegt de bal niet. Dit wil wel eens gebeuren als de afstand tussen de twee computers te groot is of de verbinding slecht is. Als je spel complexer wordt moeten er meer waardes verzonden worden om het spel goed te laten lopen. Als er elke step

veel waardes veranderen moet er veel informatie verzonden worden. Dit kan veel tijd kosten. Je spel wordt langzamer en je spel raakt uit de synchronisatie.

Een eerste manier om de communicatie van gedeelde data een stuk sneller te maken is het niet langer gebruiken van gegarandeerde verbinding. Dit kun je instellen met de volgende functie:

- `mplay_data_mode(guar)` stelt in of er wel of geen gegarandeerde verbinding gebruikt moet worden. `guar` moet true (standaard) of false zijn.

Een betere techniek om dit probleem te verhelpen heet dead-reckoning. Als je dit gebruikt Wordt de informatie van tijd tot tijd verzonden. Het spel zelf raad wat er gebeurt wat hij baseert op de informatie die het spel heeft.

We gaan dit nu gebruiken voor ons pong spel. In plaats van het verzenden van alleen de x en y positie verzenden we nu ook informatie over de speed en direction. De slave kan nu het meeste zelf berekenen. Zolang er geen nieuwe informatie van de master arriveert, kan de slave zelf berekenen waar de bal is.

We gaan nu geen gedeelde data gebruiken. In plaats daarvan gaan we berichten gebruiken. We gebruiken berichten die een verandering van de positie van de bal, een verandering van de snelheid van de bal, bat positie, etc. aangeeft. De master verzendt zulke berichten als er iets veranderd. Het controller object in de slave luistert hiernaar en veranderd de goede variabelen. Als er niks binnenkomt gebruikt de slave wat hij al weet om de bal van plaats te veranderen. Als de slave een vergissing maakt wordt het gecorrigeerd door een later bericht van de master. Als voorbeeld zetten we de volgende code in het step event van de bal:

```
{
  if (!global.master) exit;
  mplay_message_send(0,11,x);
  mplay_message_send(0,12,y);
  mplay_message_send(0,13,speed);
  mplay_message_send(0,14,direction);
}
```

In het step event van de controller object hebben we de volgende code:

```
{
  while (mplay_message_receive(0))
  {
    messid = mplay_message_id();
    val = mplay_message_value();
    // Checkt voor verandering van de bat
    if (messid == 1) bat_links.y = val;
    if (messid == 2) bat_rechts.y = val;
    // Checkt voor verandering van de bal
    if (messid == 11) object_bal.x = val;
    if (messid == 12) object_bal.y = val;
    if (messid == 13) object_bal.speed = val;
    if (messid == 14) object_bal.direction = val;
    // Checkt voor geluiden
    if (messid  == 100) sound_play(val);
  }
}
```

De berichten hoeven niet in gegarandeerde modus verzonden te worden. Als we er een missen wordt het even later wel weer gecorrigeerd. Je kan dit aangepaste spel vinden in het bestand: `pong2.gmk`.

Nu wordt je teleurgesteld waneer je het spel pong2 speelt. Het spel is soms schokkerig. Hoe komt dat? De reden is dat de verbinding traag is. Dit betekent dat de slave berichten ontvangt die een tijdje geleden werden teruggestuurd. Dit geeft als resultaat dat de bal een beetje terugschiet als het een nieuw bericht ontvangt en dan weer vooruit gaat. Laten we het voor een derde keer proberen, je kunt deze poging vinden in het bestand: `pong3.gmk`.

In deze versie wordt er alleen maar informatie uitgewisseld als de bal een batje raakt. Dus de rest van de beweging wordt gedaan door ‘dead-reckoning’. De master zorgt voor de kant van het batje van de master en de slave zorgt voor de andere kant. Als de bal van de ene kant naar de andere kant gaat worden er geen berichten meer uitgewisseld.

Zoals je ziet schokt het spel niet meer. Alleen wanneer de bal een batje raakt kan er een schokje komen. Ook kan de bal iets eerder de andere kant opgaan voordat het het batje van de tegenstander raakt. Dat komt omdat het mechanisme erop is gerekend dat beide spellen precies even snel gaan. Als één van de computers iets langzamer is kan dit problemen opleveren. Maar een schokje bij het batje is veel beter dan schokken als de bal beweegt.
Om het schokken bij het batje te voorkomen zijn ingewikkeldere mechanismen nodig. Je kunt bijvoorbeeld tijd informatie naar de andere computer sturen zodat beide zijden weten hoe snel het spel gaat bij de ander, hier kan hij zich dan op aanpassen.

Hopelijk snap je nu hoe moeilijk het synchroniseren is. Je kunt je nu misschien ook wel indenken hoe moeilijk het is voor commerciële spellen om dit te behalen. De makers komen dezelfde problemen tegen.

Een chat programma

Voor onze tweede demo maken we een klein chat programmaatje. We zullen geen limiet hebben voor het aantal spelers. We zullen ook verschillende sessies toestaan en de speler laten kiezen tussen deze sessies. We zullen berichten met strings gebruiken om de getypte tekst door te sturen.

We zullen een iets ingewikkelder mechanisme gebruiken om een connectie te maken. De eerste room heeft nu vier keuzes voor de vier verschillende connecties, maar het maakt de connecties niet. In plaats daarvan maakt het alleen een global variabele `connectietype`. De speler kan in de tweede room weer kiezen of hij een chatbox wil hosten of hij er één wil joinen. Alleen nu wordt de connectie geïnitialiseerd. Als de speler een spel host krijgt hij daarna andere vragen dan wanneer hij er eentje joint.

Nadat de connectie goed is afgelopen, wordt de sessie gemaakt of gejoined. De speler wordt daarna gevraagd zijn/haar naam te geven zodat de spelers kunnen worden onderscheiden. Het deel om te joinen is deze keer een beetje moeilijker. We maken een menu van alle verschillende sessies die aanwezig zijn, de speler kan hier tussen kiezen.

Normaal, wanneer het spel dat de sessie maakte eindigde, eindigde de sessie. Met een chat programma is dat niet wat je wilt. De andere spelers willen wel door kunnen chatten. Dit kan worden veranderd met de volgende functie:

- `mplay_session_mode(move)` stelt in of de sessie naar een andere 'host' moet worden verplaatst als de eerste 'host' stopt. `move` moet true of false zijn (false is normaal).

Het hele mechanisme in de eerste twee rooms wil je waarschijnlijk ook gebruiken voor andere spellen, want het is meestal ongeveer hetzelfde.

Als je dit hebt gehad ga je naar de chatbox room. Er is maar één controller object dat al het werk doet. Ik zal geen uitleg geven hoe je een speler iets laat typen waarna je het laat zien. Je kunt dit allemaal zien in het script en je mag dit ook gebruiken als je dat wilt. Het is allemaal best logisch (als je met GML om kan gaan). Het enige dat interessant is, is dat wanneer een speler op enter drukt, de getypte regel naar alle andere spelers wordt gestuurd met de spelers naam ervoor. Wanneer een speler joint of stopt stuurt hij een message naar alle spelers om te laten zien dat hij/zij er komt of weggaat.

Bekijk het bestand `chat.gmk` voor de details.

Conclusie

De multiplayer functies in *Game Maker* maken het mogelijk om multiplayer spellen te maken. De functies zullen je alleen maar helpen met kleine communicaties. Je zult zelf een communicatie mechanisme moeten maken. Dit is erg moeilijk. Je zult het moeten maken terwijl je het spel maakt. Het is erg moeilijk om goede multiplayer mogelijkheid nog later toe te voegen. Hier zijn wat globale richtlijnen:

- Voor de meeste simpele spellen is het het makkelijkst om het master-slave mechanisme te maken
- Zoek goed uit wie er verantwoordelijk is voor bepaalde informatie.
- Gebruik dead-reckoning waar mogelijk
- Probeer om zo min mogelijk gebruik te maken van gegarandeerde communicatie

Misschien wil je hulp vragen op ons forum of onze website http://www.yoyogames.com voor meer ideeën en technieken.