MEMBANGUN APLIKASI *GAME* SOKOBAN BERORIENTASI OBJEK PADA PONSEL DENGAN TEKNOLOGI *JAVA 2 MICRO EDITION* (J2ME)

Wahyu Pujiyono[1], Mursid Wahyu Hananto[2], Suheri[3]
[1,3]Program Studi Teknik Informatika, Universitas Ahmad Dahlan
[2]Program Studi Ilmu Komputer, Universitas Ahmad Dahlan
Kampus III UAD Jl. Prof. Dr. Soepomo, Janturan, Umbulharjo, Yogyakarta
Telp. (0274) 379418-381523 psw 104, Fax 0274-381523
e-mail: yywahyup@yahoo.com

Abstrak

Perkembangan teknologi saat ini sangatlah pesat, salah satunya adalah dibidang hiburan yaitu game. Hadirnya teknologi Java (J2ME) pada ponsel, menjadikan game tidak lagi hanya dijalankan atau dimainkan pada platform konsol maupun komputer, melainkan juga dapat dijalankan pada perangkat handheld seperti ponsel. Pengembangan game pada ponsel dengan prosessor, memori, layar dan keypad yang terbatas membutuhkan inovasi atau ide-ide kreatif dalam memodifikasi game, sehingga dapat dimainkan di ponsel dengan baik. Salah satunya adalah dengan menerapkan teknik pemrograman berorientasi objek. Permainan (game) yang banyak dimainkan di antaranya adalah yang ber-genre puzzle games. Makalah ini akan memaparkan teknik pengembangan aplikasi game sokoban yang berorientasi objek pada ponsel.

Kata kunci: *Game, Ponsel, Puzzle, Sokoban*

1. PENDAHULUAN

Perkembangan teknologi di bidang informasi maupun telekomunikasi sangatlah pesat. Hal ini ditandai dengan banyaknya perangkat telekomunikasi seperti ponsel yang beredar di masyarakat. Ponsel yang dulunya berfungsi sebagai alat untuk komunikasi, saat ini juga dapat digunakan sabagai alat untuk menikmati berbagai hiburan. Salah satu fasilitas hiburan yang dapat dinikmati adalah *game*. Salah satu jenis *game* yang terus berkembang adalah jenis *puzzle game,* misalnya Sokoban. Hal ini memberikan peluang dan tantangan bagi *game developer* untuk mengembangkan perangkat lunak *game* pada ponsel yang memiliki keterbatasan pada prosessor, memori, layar dan *keypad* dibandingkan pada PC (*Personal Computer*).

Saat ini, kecenderungan pengembangan perangkat lunak mengikuti paradigma berorientasi objek. Pengembangan perangkat lunak yang berorientasi objek memungkinkan masa pengembangan yang singkat dengan kualitas yang tinggi. Terus berkembangnya berbagai jenis permainan (*game*) khususnya jenis *puzzle games* dan besarnya peluang yang ada dalam bidang membangun aplikasi permainan atau *game* pada ponsel. Pengembangan *game* pada ponsel dengan prosessor, memori, layar dan *keypad* yang terbatas membutuhkan inovasi atau ide-ide kreatif dalam modifikasi *game*, sehingga dapat dimainkan di ponsel dengan baik.

Terdapat banyak program permainan komersil yang beredar di pasaran dengan segala keunggulan yang ditawarkan. Program permainan cukup berkembang pesat dan dikelompokkan ke dalam beberapa jenis antara lain:

a) *Role Playing Games* (RPG): merupakan program permainan *action* atau pertempuran melawan musuh, misalnya *Final Fantasy*, *Ragnarok*, dan lain-lain.
b) *First Person Shooter* (FPS): merupakan program permainan tembak-tembakan, misalnya DOOM, Quake, Unreal Tournament, Counter Strike, dan lain-lain. Khususnya untuk *game* DOOM-3 telah dinyatakan sebagai *game* terbaik.
c) *Real Time Strategy* (RTS): merupakan program permainan dengan waktu real time/tidak ada waktu tunggu, misalnya *Ages of Empire*, *Warcraft*, dan lain-lain.

d) *Turn Base Strategy* (TBS): merupakan program permainan strategi dengan sistem giliran dalam bermain, misalnya Catur, *Monopoly*, *Heroes*, *Total War*, dan lain-lain.
e) *Sport Action*: merupakan program permainan *action* berbasis olah raga, misalnya *FIFA Soccer*, *NBA Action*, dan lain-lain.
f) *Simulator Games*: permainan komputer modern kelas berat baik dari segi cara memainkannya maupun teknik pembuatannya. Permainan ini merupakan gabungan dari permainan strategi, *arcade*, dan petualangan tetapi mensimulasikan keadaan seperti sesungguhnya. Permainan ini sulit untuk dipelajari, misalnya Falcon, F-19, dan lain-lain. [10]

Game Sokoban telah diciptakan pada awal tahun 1980 oleh *Thingking Rabbit*, suatu perusahaan *game* komputer di kota *Takarazuka*, *Jepang*. Desain *game*-nya mendapat kemenangan untuk juara pertama pada suatu kontes *games* komputer. Karena dari kesederhanaan dan elegan dari *rules*-nya serta kerumitan intelektual yang menantang dari masalah-masalah yang disusun, Sokoban sangat cepat menjadi suatu hiburan yang populer.

Beberapa versi dari *game* Sokoban diciptakan kurang lebih setahun ini, di antaranya pada versi PC, Macintosh dan Unix. Terdapat suatu set standar dari 50 masalah, dimulai dari yang mudah sekali sampai yang sangat sulit untuk seseorang menyelesaikannya. Menurut Hiramatsu, sekelompok dari 50 masalah ini dihasilkan *dari* suatu versi PC oleh ***Spectrum Holobyte*** tahun 1984. Konfigurasi masalah yang sangat mirip dapat ditemukan pada "Sokoban 2" tahun 1984 dan sekarang termasuk dalam koleksi "*Sokoban Perfect*". Beberapa dari masalah telah diubah menjadi lebih ramping, mungkin dikecilkan ke dalam suatu format 19x16.

Desain *yang* menarik, ide dalam penciptaannya dan solusi yang berbelit-belit *dalam* menyelesaikannya membuat Sokoban menjadi suatu permainan yang intelektual. Fakta-fakta apa yang membuat Sokoban menarik untuk dimainkan? [6]

a). Sokoban mudah untuk dimengerti dan visualis. Jika ingin melihat dimana pencarian membuat kesalahan atau dimana pencarian sangat efisien, Sokoban sangat mudah untuk saya memahaminya seperti suatu daerah atau lapangan.
b). Digunakan sebagai tes kasus.
c). Perbedaan tingkat-tingkat kesulitan. Tingkat-tingkat kesulitan dalam menyelesaikan permasalahan Sokoban dimulai dari yang paling mudah sampai dengan yang paling sulit. Dari tingkat yang lebih mudah, pemain dapat meningkatkan kemampuannya pada permainan dengan tingkat yang lebih sulit.
d). Perbedaan tipe-tipe masalah. Tidak hanya ukuran dan kesulitan tetapi terdapat varisasi dari tipe permasalahan Sokoban, sehingga dibutuhkan perbedaan konsep untuk menyelesaikannya.
e). *Deadlock*, Sokoban menawarkan suatu fitur yang menarik. Jika pemain tidak berhati-hati, maka dapat membuat konfigurasi masalah yang tak dapat diselesaikan.
f). Dan tidak hanya itu, Sokoban menyenangkan.

Aturan dalam bermain Sokoban sangatlah sederhana, yaitu pemain hanya mengontrol seorang manusia atau pendorong (*pusher*) untuk melangkah dan mendorong semua batu atau kotak sehingga menempati tempat penyimpanan (*store*) batu atau kotak dalam suatu *maze* (tempat yang penuh dengan rintangan dan berbelit-belit). Pemain hanya dapat melakukan langkah ke kiri, ke kanan, ke atas atau ke bawah. Namun pemain juga harus berhati-hati karena dapat membuat permainan tidak dapat diselesaikan atau dalam keadaan *deadlock*, sehingga untuk dapat menyelesaikannya dibutuhkan kemampuan intelektual yang cukup baik.

Pertama-tama, *player* menekan sebuah tombol untuk menjalankan *game*. Kemudian akan ditampilkan *splashscreen* yang merupakan intro dari *game*. Setelah itu, *player* dapat langsung bermain pada *level* 0 sebagai *level* pembelajaran dan dilanjutkan *level* 1 sampai dengan *level* 30.

Terdapat menu pilihan yang langsung dapat digunakan *player* ketika bermain. Menu pilihan tersebut terdiri dari menu ***restart level*** yang berfungsi untuk mengulang kembali permainan dari awal pada *level* tersebut. Menu ***level* berikutnya** yang berfungsi untuk langsung menuju pada *level* satu tingkat di atasnya. Menu ***level* sebelumnya** yang berfungsi untuk langsung menuju pada *level* satu tingkat di bawahnya. Menu **pilih *level*** berfungsi untuk langsung menuju pada *level* yang diinginkan. Menu **lihat *score*** berfungsi untuk menampilkan *score*/nilai yang terbaik. Menu ***about*** berfungsi untuk menampilkan informasi singkat tentang pengembang. Menu **undo** yang berfungsi untuk kembali pada langkah sebelumnya dalam bermain. Menu **keluar** yang berfungsi untuk mengakhiri permainan.

Pada tiap-tiap *level*, terdiri dari *wall*, *store*, *packet*, *pusher* dan *ground*. Tujuan dari permainan ini adalah *player* harus meletakkan semua *packet* pada *store* dengan cara menggerakkan *pusher* untuk memindahkan *packet-packet* yang ada ke *store*.

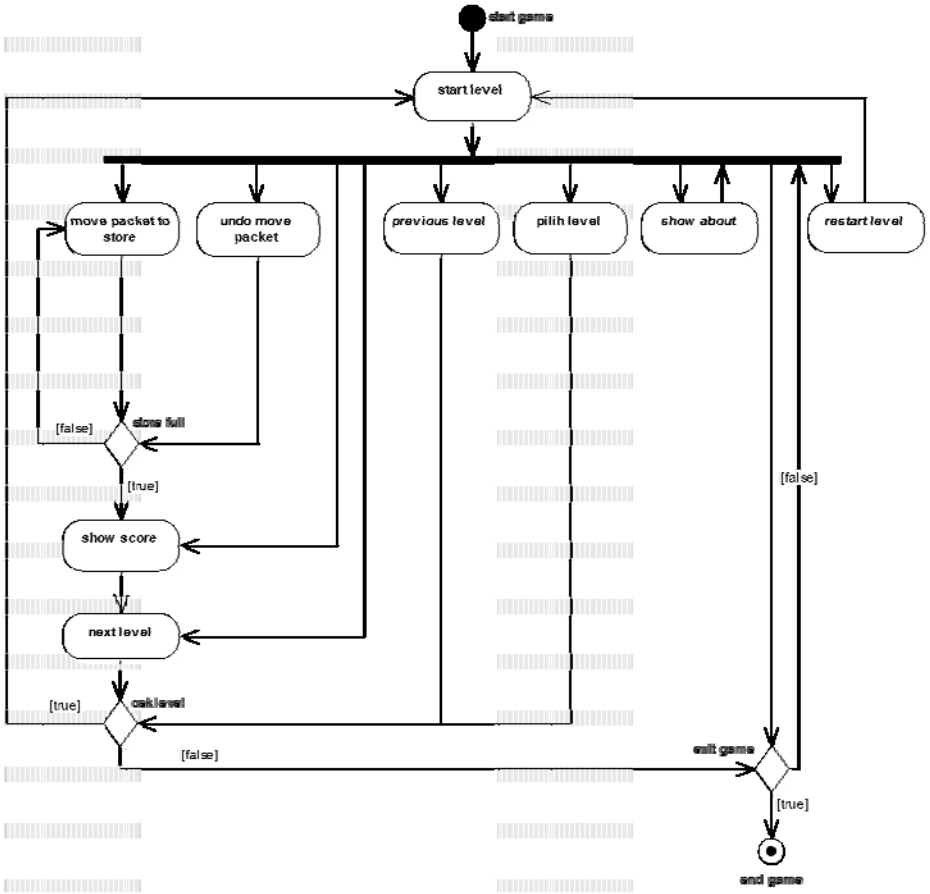
2. PERANCANGAN

2.1. Analisis dan Pemodelan Kebutuhan Sistem

Tahap-tahap analisis dan pemodelan kebutuhan yang dilakukan dalam pengembangan sistem berorientasi objek adalah sebagai berikut:

a). Aktivitas Bermain *Game* Sokoban

Tahap ini dilakukan untuk mendeskripsikan aktivitas yang dilakukan pemain (*player*) pada saat bermain *game* sokoban. Aktivitas ini berguna untuk melakukan pemodelan kebutuhan.



Gambar 1. *Activity diagram game* sokoban

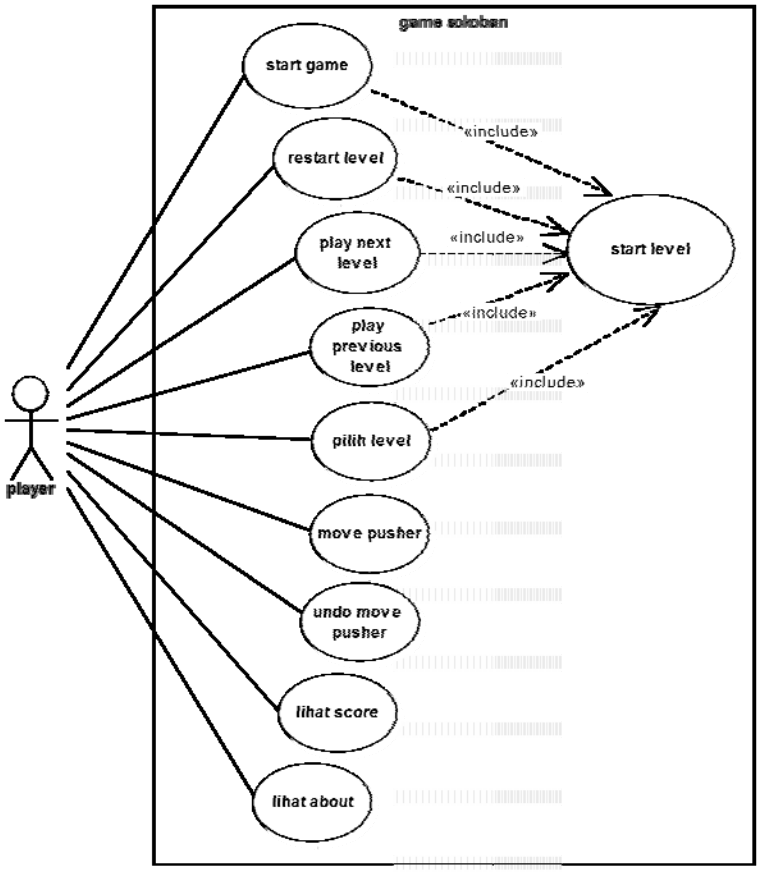
Berdasarkan aktivitas *player* dalam bermain *game* sokoban di atas, diperoleh kebutuhan yang harus dipenuhi oleh sistem seperti ditunjukkan pada Tabel 1.

b). Pemodelan Kebutuhan dengan *Use Case Diagram*

Dari kebutuhan *user* seperti ditunjukkan pada Tabel 1, diperoleh *actor* yang dimungkinkan menjadi *user* sistem yang akan dikembangkan dan *use case* dari masing-masing *actor*. Dalam hal ini, *actor* tersebut adalah *player*. *Player* merupakan satu-satunya *actor* yang berinteraksi dengan sistem. *Player* adalah pemain yang memainkan *game* sokoban. *Use case diagram* untuk *actor player* ditunjukkan pada Gambar 2.

Tabel 1. Tabel kebutuhan user (*user requirement*)

No	Requirement	Aktor	*Use Case*
1	*Player* memulai *game* sokoban	*Player*	*Start game*
2	*Player* langsung bermain pada *level* yang ada	*Player*	*Start level*
3	*Player* bermain dengan cara menggerakkan *pusher* untuk memindahkan *packet-paket* ke *store*	*Player*	*Move pusher*
4	Jika *player* salah dalam melangkah, *player* bisa kembali pada posisi sebelumnya	*Player*	*Undo move pusher*
5	*Player* bisa memainkan kembali permainan dari awal pada *level* tersebut	*Player*	*Restart level*
6	Jika semua *store* telah ditempati oleh *packet-packet*, maka permainan pada *level* tersebut berakhir dan *player* bisa melihat *score*/nilai permainan	*Player*	Lihat *score*
7	*Player* bermain pada *level* selanjutnya	*Player*	*Play next level*
8	*Player* bisa memainkan kembali permainan pada *level* sebelumnya	*Player*	*Play previous level*
9	*Player* bisa memilih *level* permainan yang akan dimainkan	*Player*	Pilih *level*
10	*Player* bisa melihat informasi singkat tentang pengembang	*Player*	Lihat *about*



Gambar 2. *Use case diagram* untuk *actor player*

2.2. Perancangan Sistem

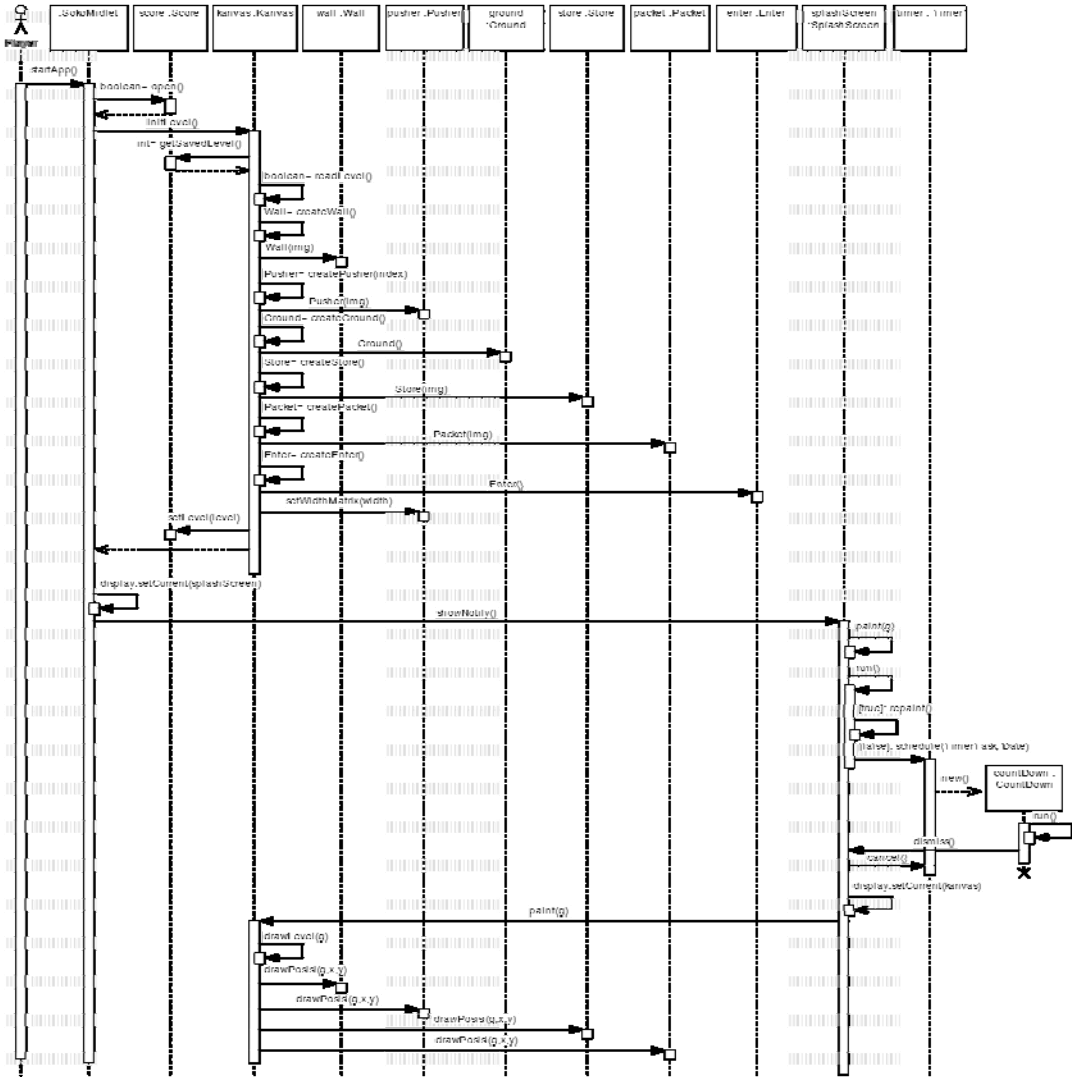
Berikut ini adalah *sequence diagram* yang menunjukkan urutan proses yang terjadi pada masing-masing *use case* :

a). *Use Case Start Game*

Urutan proses dari *use case Start Game* adalah sebagai berikut:

1) Pada saat *game* dijalankan, mula-mula objek *MIDlet* membuka media penyimpanan dari objek *score*. Jika media penyimpanan (*Record Store*) tidak ditemukan maka media penyimpanan akan diciptakan.
2) Selanjutnya objek *kanvas* melakukan inisialisasi *level* yang akan dimainkan.
3) Inisialisasi *level* dilakukan dengan mengambil data *level* yang pernah tersimpan pada media penyimpanan.
4) Selanjutnya objek *kanvas* membaca *level* yang telah diperoleh dan menciptakan objek-objek seperti *wall*, *pusher*, *ground*, *store*, *packet*, dan *enter*.
5) Objek *kanvas* mengirim pesan data *level* ke objek *score* agar melakukan penyimpanan.
6) Selanjutnya objek *MIDlet* menampilkan objek *splashscreen* dan menciptakan objek *CountDown* sebagai *timer* untuk menghitung mundur waktu. Jika waktu telah habis maka dilanjutkan dengan menampilkan objek *kanvas*.
7) Objek *kanvas* meneruskan dengan menampilkan objek-objek hasil pembacaan *level*, yaitu *wall*, *pusher*, *store* dan *packet*.

Berdasarkan urutan proses di atas, Gambar 3 berikut adalah *sequence diagram* untuk *use case start game*.



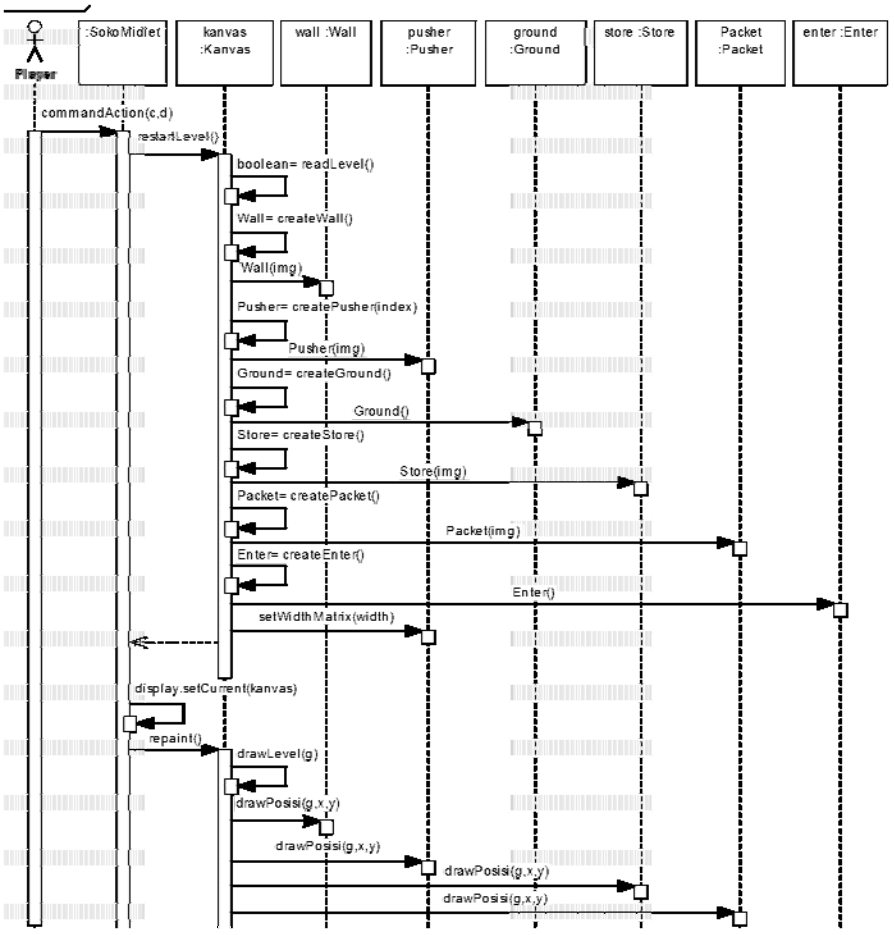
Gambar 3. *Sequence diagram* dari *use case start game*

b). *Use Case Restart Level*

Urutan proses dari *use case restart level* adalah sebagai berikut:

1) *Player* mengirim pesan untuk melakukan *restart level* kepada objek *MIDlet*.
2) Objek *MIDlet* meneruskan pesan tersebut kepada objek *kanvas* untuk diproses lebih lanjut.
3) Selanjutnya objek *kanvas* membaca *level* yang telah diperoleh dan menciptakan objek-objek seperti *wall*, *pusher*, *ground*, *store*, *packet*, dan *enter*.
4) Objek *kanvas* meneruskan dengan menampilkan objek-objek hasil pembacaan *level*, yaitu *wall*, *pusher*, *store* dan *packet*.

Berdasarkan urutan proses di atas, Gambar 4 berikut adalah *sequence diagram* untuk *use case restart level.*



Gambar 4. *Sequence diagram* dari *use case restart level*

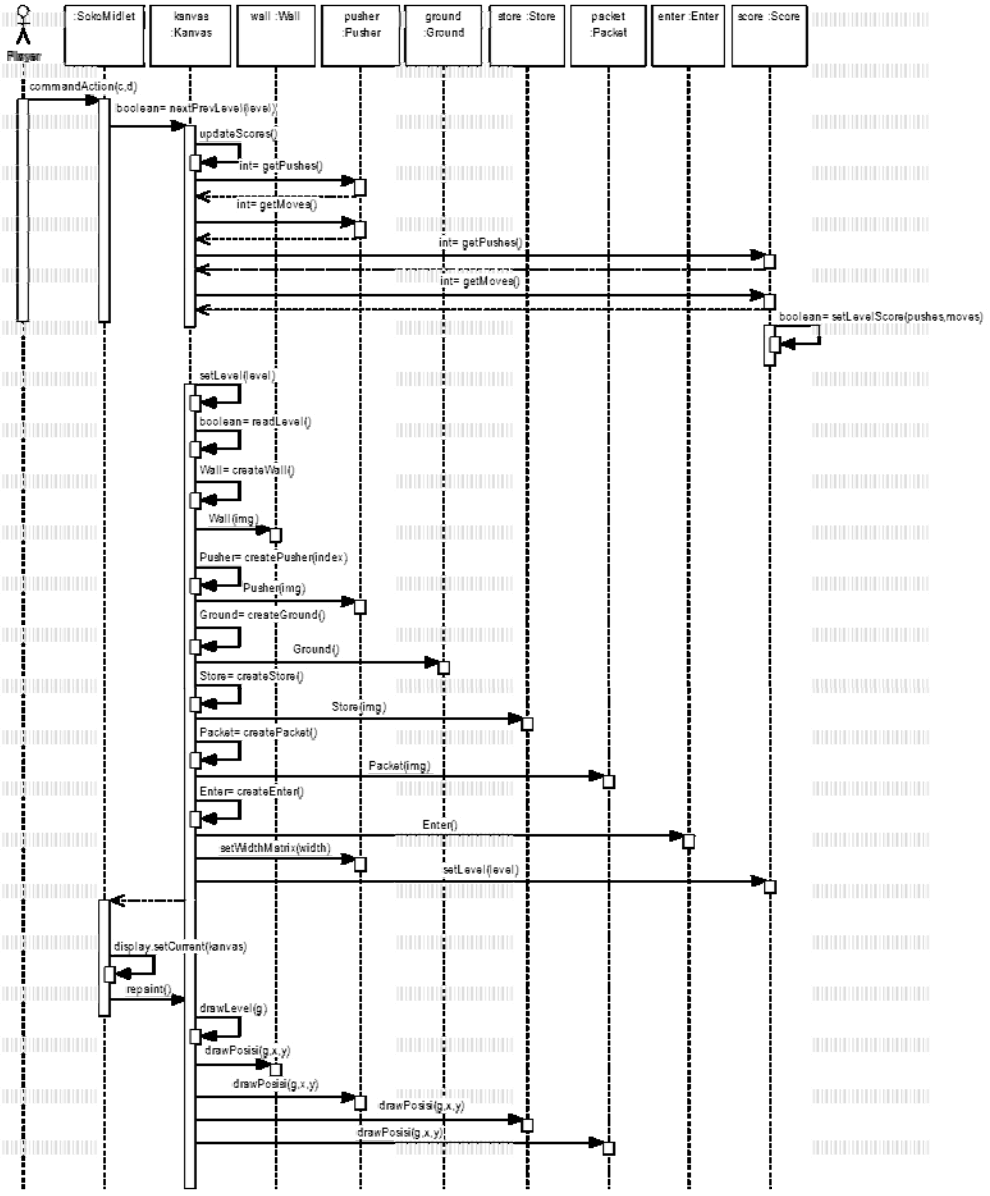
c). *Use Case Play Next Level*

Urutan proses dari *use case play next level* adalah sebagai berikut :

a) Setelah *player* menyelesaikan permainan, kemudian dilanjutkan dengan memainkan permainan pada *level* selanjutnya atau *player* mengirim pesan untuk melakukan *play next level* kepada objek *MIDlet*.
b) Objek *MIDlet* meneruskan pesan kepada objek *kanvas*, kemudian objek *kanvas* melakukan *update* nilai atau *score* permainan.
c) *Update* nilai dilakukan oleh objek *kanvas* dengan mengirim pesan kepada objek *pusher* untuk memperoleh data berupa banyaknya *pushes* (mendorong *packet*) dan banyaknya *moves* atau langkah yang telah dilakukan dari setiap *level* yang dimainkan.
d) Kemudian objek *kanvas* juga mengirim pesan ke objek *score* untuk memperoleh data *pushes* dan *moves* dari *level* tersebut yang telah disimpan dan selanjutnya menyimpan data tersebut.

e) Selanjutnya objek *kanvas* melakukan set *level* berikutnya yang diikuti dengan membaca *level* tersebut. Setelah *level* dibaca, maka diciptakanlah objek-objek seperti *wall*, *pusher*, *ground*, *store*, *packet*, dan *enter*.
f) Objek *kanvas* mengirim pesan kepada objek *score* untuk melakukan penyimpanan data *score* pada *level* tersebut.
g) Kemudian objek *MIDlet* menampilkan objek *kanvas*, objek *kanvas* meneruskannya dengan menampilkan objek-objek hasil pembacaan *level*, yaitu *wall*, *pusher*, *store* dan *packet*.

Berdasarkan urutan proses di atas, Gambar 5 berikut adalah *sequence diagram* untuk *use case play next level.*



Gambar 5. *Sequence diagram* dari *use case play next level*

d). *Use Case Move Pusher*

Urutan proses dari *use case move pusher* adalah sebagai berikut:

1) *Player* mengirim pesan kepada objek *kanvas* dengan menekan sebuah tombol yang selanjutnya diterjemahkan menjadi nilai yang akan menentukan pergerakan dari *pusher*, yaitu ke kiri, ke kanan, ke atas atau ke bawah.

2) Selanjutnya nilai tersebut dikirim ke objek *pusher* untuk diproses lebih lanjut.
3) Kemudian objek *pusher* akan melakukan beberapa pengecekan, yaitu:
   (a). Apakah *pushes* yang dilakukan menyebabkan objek *packet* yang telah tersimpan pada objek *store* berpindah tempat ke objek *ground*. Jika ya, maka nilai *pushes* akan dikurangi. Jika tidak, tidak ada aksi yang dilakukan.
   (b). Apakah tujuan melangkah terdapat objek *wall*. Jika ya, maka posisi dari objek *pusher* tidak berubah.
   (c). Apakah tujuan melangkah terdapat objek *packet*. Jika ya, maka dilakukan dua pengecekan lagi, yaitu apakah tujuan berikutnya setelah objek *packet* terdapat objek *wall* atau *packet*. Jika ya, maka posisi dari objek *pusher* dan *packet* tidak berubah. Jika pengecekan pertama tidak, maka objek *pusher* mengirim pesan kepada objek *packet* untuk melangkah. Dalam hal ini, objek *packet* juga melakukan beberapa pengecekan, yaitu :
      (1) Apakah posisi objek *packet* berada pada objek *store*. Jika ya, maka posisi *packet* akan berpindah sesuai dengan tujuan. Kemudian dilakukan pengecekan apakah tujuannya terdapat objek *store*. Jika ya, maka akan dihitung objek *packet* yang telah berhasil dipindahkan ke objek *store*. Dari pengecekan pertama jika hasilnya tidak, maka dihitung objek *packet* yang telah berhasil dipindahkan ke objek *store*.
      (2) Apakah tujuannya terdapat objek *store*. Jika ya, maka posisi objek *packet* akan berpindah dan dihitung kembali banyaknya objek *packet* yang telah berhasil dipindahkan ke objek *store*.
   (d). Jika semua pengecekan kondisi di atas tidak terpenuhi, maka posisi dari objek *pusher* dan *packet* berpindah sesuai tujuannya dan dilakukan proses penyimpanan langkah.
4) Jika semua pengecekan kondisi di atas tidak terpenuhi, maka posisi dari objek *pusher* akan berpindah sesuai denga tujuannya dan dilakukan proses penyimpanan langkah.
5) Kemudian objek *kanvas* akan menggambarkan ulang semua objek (*wall*, *pusher*, *store*, dan *packet*) sesuai dengan posisi masing-masing.
6) Selanjutnya objek *kanvas* melakukan pengecekan apakah semua objek *packet* telah berhasil diletakkan pada objek *store*. Jika ya, maka *score* atau nilai akan ditampilkan.

Sequence diagram dari *use case move pusher* ditunjukkan pada Gambar 6.

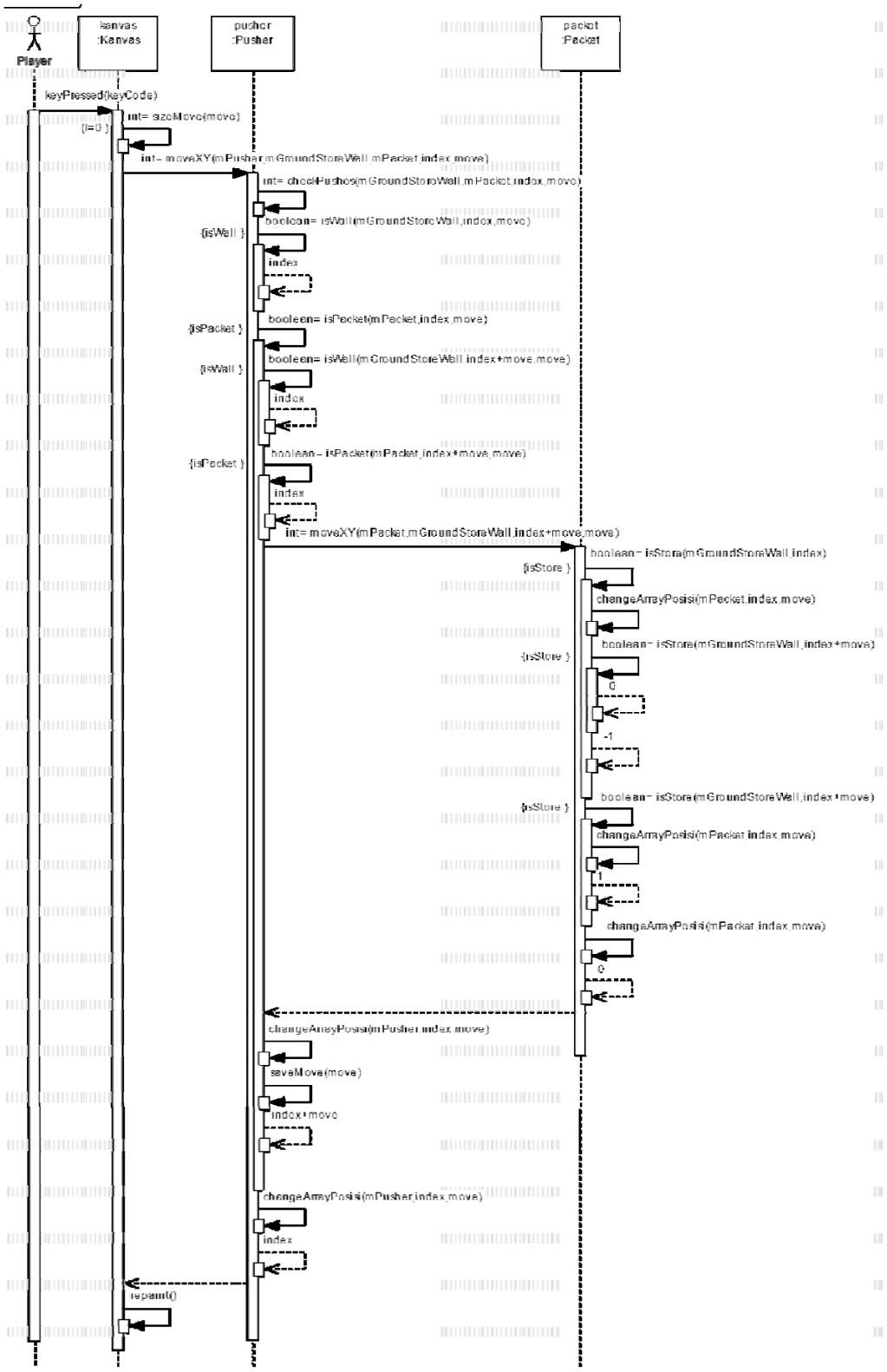
e). *Use Case Undo Move Pusher*

Urutan proses dari *use case undo move pusher* hampir sama dengan urutan proses dari *use case move pusher*, hanya saja proses pada *use case undo move* tidak dilakukan proses penyimpanan langkah. Sehingga urutan prosesnya tidak perlu dijelaskan lagi. Gambar 7 adalah *sequence diagram* untuk *use case undo move pusher*.

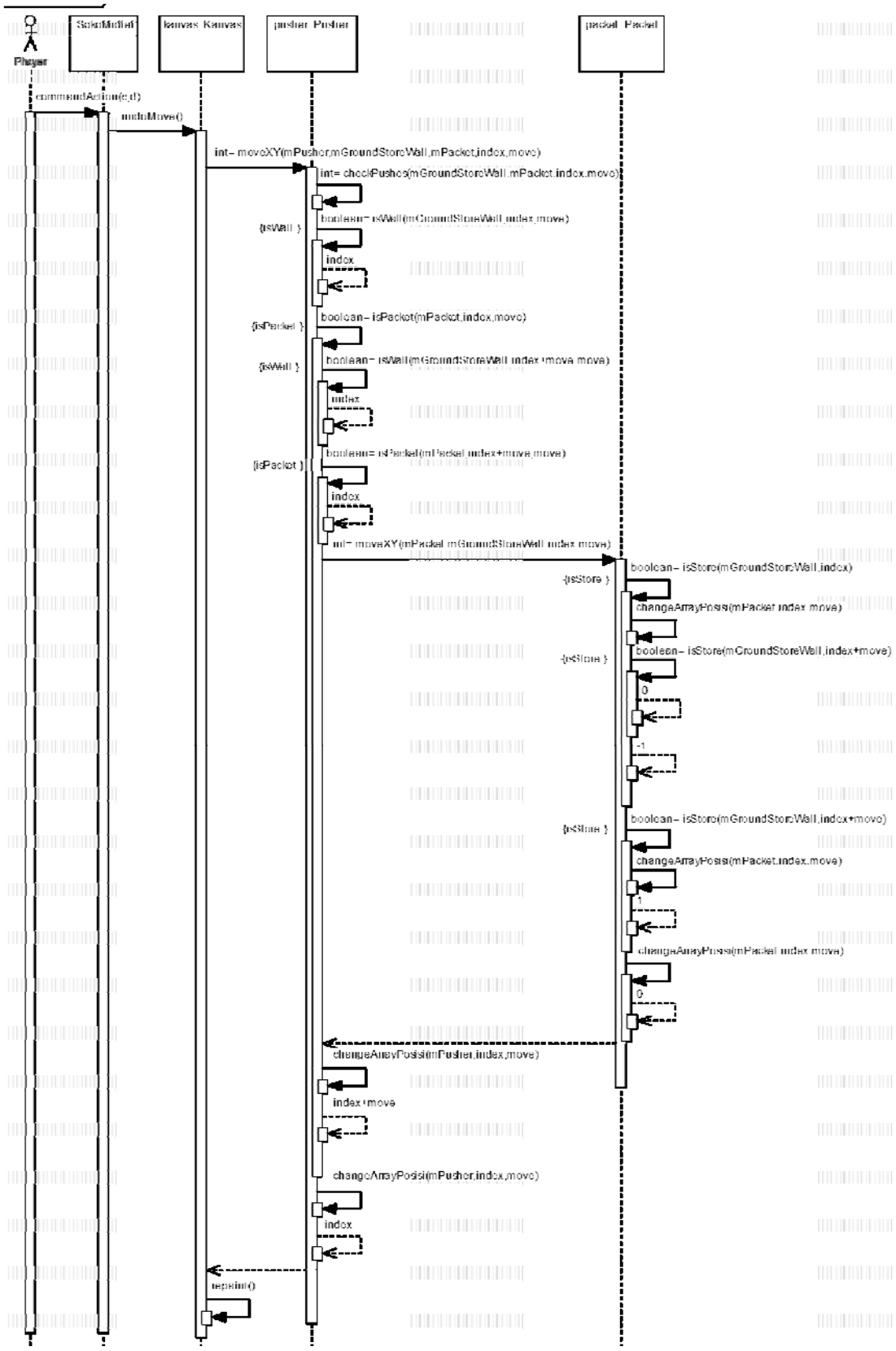
Berdasarkan diagram-diagram *sequence*, diperoleh objek-objek yaitu *sokomidlet*, *splashscreen*, *kanvas*, *wall*, *pusher*, *ground*, *store*, *packet*, *enter*, *countdown*, dan *timer*. Masing-masing objek ini menjadi *class-class* yang dibutuhkan, yaitu *class SokoMidlet* sebagai *class* induk, *SplashScreen*, *Kanvas*, *Wall*, *Pusher*, *Ground*, *Store*, *Packet*, dan *Enter*, sedangkan objek *countdown* akan menjadi *class* yang berada di dalam *class SplashScreen* dan objek *timer* tidak dijadikan sebagai *class* baru karena sudah disediakan oleh Java. Dari beberapa objek, terdapat kemiripan pada atribut dan *method* yang dimiliki beberapa *class*, yaitu *class wall*, pusher, ground, store, packet, dan enter. Sehingga dibentuk *class* baru yaitu *GameObject*. Gambar 8 adalah *diagram class* dari *class-class* tersebut di atas.

3. HASIL DAN PEMBAHASAN

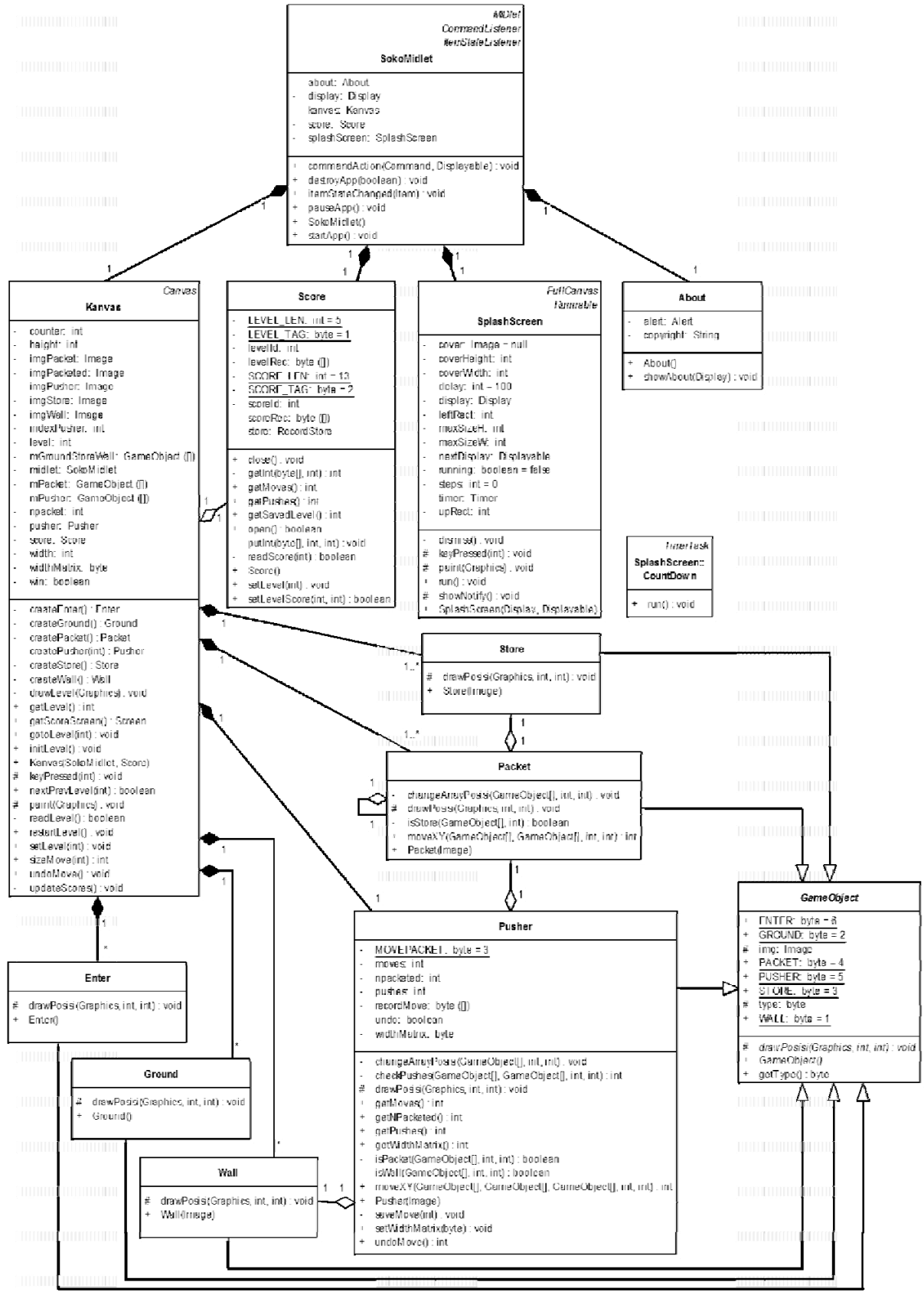
Menu screen (Gambar 9) merupakan menu yang digunakan untuk melakukan aksi-aksi tertentu, yaitu: "*restart Lvl*" berfungsi untuk mengulang kembali permainan dari awal pada *level* tersebut, "*Lvl* berikutnya" berfungsi untuk bermain pada *level* berikutnya, "*Lvl* Sebelumnya" berfungsi untuk bermain pada *level* sebelumnya, "Pilih *Level*" yang berfungsi untuk bermain pada *level* yang diinginkan, "Lihat *Score*" yang berfungsi untuk melihat *score* atau nilai permainan pada *level* tersebut, "*About*" berfungsi untuk melihat informasi singkat tentang pengembang. Tampilan dari masing-masing menu dapat ditunjukkan pada Gambar 10 dan 11.



Gambar 6. *Sequence diagram* dari *use case move pusher*

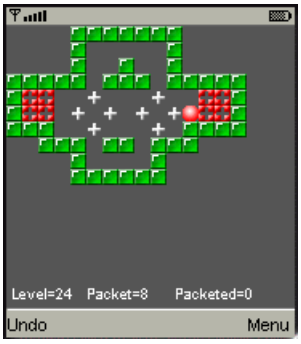


Gambar 7. *Sequence diagram* dari *use case undo move pusher*



Gambar 8. *Class diagram game* sokoban

Gambar 9. Tampilan *Menu Screen*



Gambar 10. Tampilan *Level* Permainan dan Pilih *Level*

Gambar 11. Tampilan *Score*

4. SIMPULAN

Telah dapat dikembangkan sebuah aplikasi permainan (*game*) Sokoban ber-*genre puzzle games* yang menarik menggunakan *Java 2 Micro Edition* dengan MIDP 1.0 dan terdiri dari 30 level permainan yang bisa dijalankan pada komputer dengan menggunakan emulator ponsel dan telah diimplementasikan langsung pada ponsel-ponsel yang telah mendukung teknologi *Java*. Permainan (*game*) Sokoban ini dapat dijadikan sarana hiburan yang dapat menguji konsentrasi dan cara berfikir sistematis serta menguji daya ingat sesorang yang menyenangkan.

DAFTAR PUSTAKA

[1] Fachrisal, A., Haryo, B., Auw, J., Arnawa, W., Tommy, P., Tino, D.P., Antony, S., Fery, Michael, E., Edisi 07, hal 57, **"Tip & Trik Mengoptimalkan Sistem Operasi Windows Untuk Games"**, Play/Level Majalah Game PC Internasional, PT. Elex Media Komputindo, Jakarta, 2002.

[2] Hartanto, A.A., **"Pemrograman Mobile Java dengan MIDP 2.0"**, PT. Elex Media Komputindo, Jakarta, 2003

[3] Sunaryo, A., **"Membangun Aplikasi Game Mover Pada Ponsel Menggunakan Java 2 Micro Edition (J2ME)"**, Skripsi S-1, Universitas Ahmad Dahlan, Yogyakarta, 2005.

[4] Wicaksono, A., **"Pemrograman Aplikasi Wireless dengan Java"**, PT. Elex Media Komputindo, Jakarta, 2002.

[5] Zaidir, **"Metode Collision dan Sistem Acak sebagai Dasar Pembuatan Game Menggunakan Microsoft Visual Basic 6.0"**, Skripsi S-1, Universias Ahmad Dahlan, Yogyakarta, 2005.

[6] http://www.cs.ualberta.ca/~games/Sokoban/

[7] http://www.java.sun.com/J2ME

[8] http://www.java.sun.com/product/midp/